VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Nástroje pro hromadné generování a předzpracování dat pro fulltextové vyhledávání včetně API

Tools for Bulk Generation and Preprocessing of Data, Including Full-text Search API

2018 Bc. Eduard Kubanda

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Eduard Kubanda**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Nástroje pro hromadné generování a předzpracování dat pro fulltextové vyhledávání včetně API
Tools for Bulk Generation and Preprocessing of Data, Including Full-text Search API

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je provést analýzu, návrh a implementaci nástroje pro hromadné generování a předzpracování dat pro fulltextové vyhledávání včetně implementace veřejného API rozhraní pro komunikaci s externími zdroji a mobilní aplikací v rámci projektu Gloffer.

1. Student analyzuje dostupné technologie se zaměřením na platformu Linux a implementační prostředí a technologie MySQL, MongoDB (Aerospike), Elastic Search (Lucene, Solr), nGinx (Apache), Redis, Rabbit MQ, PHP7 a Framework Nette. Pro účely implementace API zvolí vhodnou technologii (JAVA atd.).
2. Student vytvoří a zdokumentuje postup přidávání nových objektů do full-textového vyhledávání. Import a učení systému Gloffer novým produktovým oblastem, správa kategorií, vlastností, hodnot a vícejazyčného obsahu s ohledem na vhodnou parametrizaci při vyhledávání.
3. Student provede analýzu vhodných datových struktur a jejich mapování pro full-text index Elastic Search. Dále bude realizován systém našeptávačů pro full-textové vyhledávání.
4. Student navrhne a implementuje API pro komunikaci s mobilní aplikací a externími systémy pro import a export nabídek a poptávek.
5. V závěru student provede srovnání výsledné implementace s existujícími řešeními a zhodnotí, zda je efektivní budovat vlastní informační platformu nebo využít již existující řešení nebo cloudu.

Seznam doporučené odborné literatury:

[1] GORMLEY, Clinton a Zachary TONG. Elasticsearch: the definitive guide. ISBN 1449358543.
[2] SHKLAR, Leon. a Rich. ROSEN. Web application architecture: principles, protocols and practices. 2nd ed. Hoboken, NJ: Wiley, c2009. ISBN 047051860x.
[3] SHIVAKUMAR, Shailesh Kumar. Architecting high performing, scalable and available enterprise web applications. ISBN 9780128022580.
[4] WEERAWARANA, Sanjiva. Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more. Upper Saddle River, NJ: Prentice Hall PTR, c2005. ISBN 0131488740.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radoslav Fasuga, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikáce, z ktorých som čerpal.

V Ostrave 25. apríla 2018 .....

Chcem poďakovať Ing. Radoslav Fasuga, Ph.D. za profesionálnu pomoc, užitočné rady, ochotu a venovaný čas pri riešení tejto diplomovej práce. Veľká vďaka patrí taktiež mojej rodine a priateľom za poskytnutú oporu.

Abstrakt

Predmetom diplomovej práce je vývoj častí pre informačný systém orientovaný na produktové agregácie. Obsah práce pozostáva z implementácie komplexnej fulltextovej funkcionality, dátových procesov v súvislostí s generovaním obsahu a jeho migrácie do fulltextový štruktúr a poslednou časťou je webové aplikačné rozhranie.

Základom pre všetky časti je analýza existujúcich riešení a súčasného stavu systému. Fulltextová funkcionalita bola implementovaná pomocou technológie Elasticsearch. Dátové procesy procesy pracujú s relačnou schémou informačného systému, ale taktiež využívajú vlastnosti NoSQL databáz, konkrétne MongoDB. Webové aplikačné rozhranie bolo implementované architektonickým štýlom REST, prostredníctvom Java frameworku Spring.

Kľúčové slová: informačný systém, fulltext, Elasticsearch, vyhľadávanie, SQL, NoSQL, webová služba, aplikačné rozhranie, REST, Java, Spring

Abstract

The subject of this master thesis is the development of components within information system focused on product aggregation. The content of the work consists of the implementation of complex fulltext functionality, data processes in relation to content generation and its migration to fulltext structures and the last part is web application interface.

The foundation for all parts is the analysis of existing solutions and the current state of the system. Fulltext functionality was implemented using Elasticsearch technology. Data processes work with the relational schema of the information system, but also use the properties of NoSQL databases, namely MongoDB. The web application interface was implemented with the architectural style REST, through the Java framework Spring.

Key Words: information system, fulltext, Elasticsearch, search, SQL, NoSQL, web service, aplication interface, REST, Java, Spring

Obsah

Zoznam použitých skratiek a symbolov

IS	–	Information system
DB	–	Database
REST	–	Representational State Transfer
SQL	–	Structured Query Language
NoSQL	–	Non SQL, non relational
API	–	Application programming interface
XML	–	eXtensible Markup Language
JSON	–	JavaScript Object Notation
NRT	–	Near Real Time
HA	–	High availability
CRUD	–	Create, read, update, delete
CLI	–	Command-line interface
HTML	–	HyperText Markup Language
ASCII	–	American Standard Code for Information Interchange
DSL	–	Domain Specific Language
TD/IDF	–	Term Frequency/Inverse Document Frequency
EAN	–	European Article Number
ISBN	–	International Standard Book Number
ACID	–	Atomicity, Consistency, Isolation, Durability
JDBC	–	Java Database Connectivity
ID	–	Identifier
MVC	–	Model–view–controller
CD	–	Compact Disc
DVD	–	Digital Versatile Disc
LAN	–	Local Area Network
WAN	–	Wide Area Network
SOAP	–	Single Object Access Protocol
WSDL	–	Web Services Description Language
SSL	–	Secure Sockets Layer
TLS	–	Transport Layer Security
HMAC	–	Keyed-hash message authentication code
JWT	–	JSON Web Token
JPA	–	Java Persistence API
WAR	–	Web application Archive
GDRP	–	General Data Protection Regulation

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov zdrojového kódu

1 Úvod

Komplexnosť informačného systému značne rozvíja jeho použitie ako business nástroja. Možnosť prístupu z viacero typov zariadení a previazanosť so sofvérom tretích strán sa prejavuje v náraste riešenej domény úloh a dát.

Táto diplomová práca nadväzuje na bakalársku prácu, ktorej cieľom bolo vytvorenie informačného systému portálu Gloffer. Vyvíjaný informačný systém od vtedy prešiel výraznými zmenami, predovšetkým v business modele danej aplikácie. Spôsobené zmeny mali priamy dopad na vývoj, jeho zložitosť, ako aj na doménu a objem dát. Pôvodný informačný systém obsahoval súbor relatívne nezávislých podsystémov, ktoré cielili na rôzne domény business použitia, ako automobilový portál, dovolenkový portál a pod. Nový informačný systém predstavuje produktový agregátor, ktorého hlavnou a unikátnou vlastnosťou je funkcionalita dopytu a ponuky nad produktami. Kompetencie vývoja nového informačného systému boli rozdelené medzi frontend a backend časť. Backend predstavoval predovšetkým zabezpečenie dátovej domény produktov a operácie spojené s touto úlohou. Úlohou frontend časti z pohľadu definície je zabezpečenie zobrazovania dát a interakcia s používateľom. Z dôvodu relatívne malého vývojového týmu sa v prípade tejto aplikácie vo frontend časti riešila aj časť samotnej funkcionality informačného systému, predovšetkým funkcionality spojenej s dopytom a ponukou.

Obsah diplomovej práce pozostáva z troch hlavných úloh. Prvou je zabezpečenie fulltextovej funkcionality. Táto úloha predstavuje zabezpečenie samotnej infraštruktúry, definície dátových objektov a fulltextových dopytov. Cieľom je vývoj infraštruktúry v reálnom, produkčnom merítku. Druhou úlohou je zabezpečenie obsahu a dátových procesov pre fulltextové štruktúry. Úloha pojednáva predspracovanie obsahu a procesy migrácie dát v dátovej vrstve riešenia.

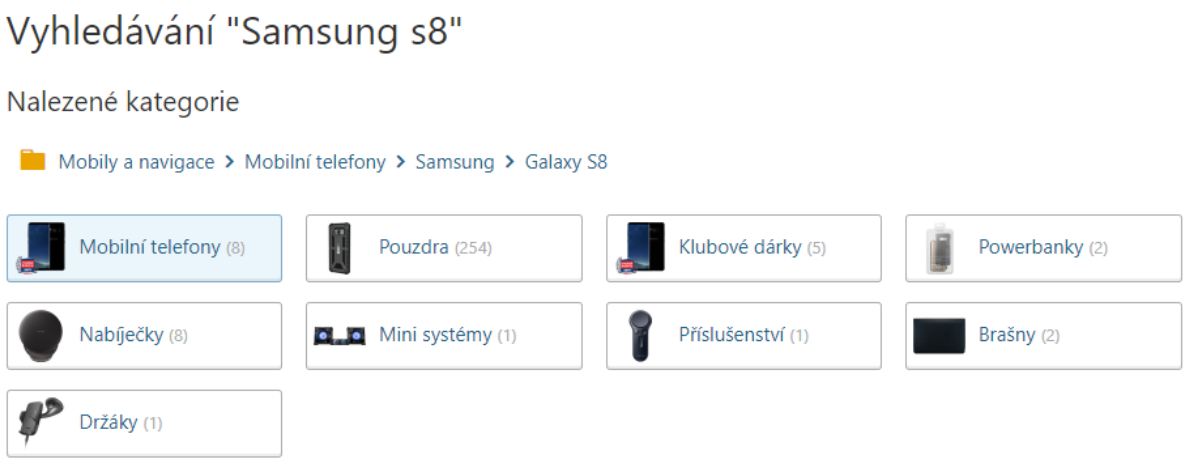
Tretia časť predstavuje vývoj aplikačného rozhrania (API) pre daný informačný systém a priamo súvisí s riešením nárastu celkovej komplexnosti. Informačný systém ako produktový agregátor s funkcionalitou dopytu a ponuky sa postupom vývoja dostal do verzie, kedy obsahoval potrebnú funkcionalitu a boli definované postupy spracovania a vykonania úloh v danej doméne. Pre vývoj API táto situácia predstavuje určitý predpis, nutný základ. Cieľom vývoja API je predovšetkým prenesenie aktuálnej logiky z klienta, frontend časti do samostatnej vrstvy, čo implicitne predstavuje novú verziu architektúry a centrálnu správu požiadaviek a odpovedí systému. Dôležitým faktom je vývoj mobilnej aplikácie, ktorá v rámci systému predstavuje ďalšieho klienta.

2 Fulltextová funkcionalita IS

Informačný systém ako celok integruje širokú škálu funkcionality a umožňuje používateľovi vykonávať úlohy v danej doméne. Za fundamentálny mechanizmus takéhoto softvéru môžeme považovať dynamický proces získania relevantných dát na základe (používateľského) vstupu. V prípade informačných systémov, ktoré sú prezentované formou webovej stránky, mobilnej aplikácie a pod., môže byť vstupom kliknutie na tlačidlo, kliknutie na odkaz, zadanie čísla stránky (stránkovací mechanizmus) a iné formy interakcie. Vstupné a výstupné dáta sú v tomto prípade jednoznačné – tlačidlo má jednoznačnú funkciu a odkaz smeruje na konkrétnu podstránku. Ak je ale dátová doména IS komplexná, je prakticky nemožné definovať exaktné vstupy a odpovedajúce výstupné dáta. Modelovým príkladom je informačný systém s predajom elektroniky. Systém môže obsahovať niekoľko tisíc produktov zaradených v desiatkach kategórií. Tieto systémy implicitne obsahujú katalóg, ktorý je možné prechádzať formou odkazov a listovať v ňom. Takýto prístup je používateľsky neprívetivý vzhľadom na čas vynaložený k získaniu požadovaných výsledkov, keďže používateľ nemusí poznať presné zaradenie produktu ktorý hľadá. Riešením pre popísaný scenár a ďalšie je fulltextové vyhľadávanie. Fulltextové vyhľadávanie je technika vyhľadávania nad dátami, pričom vstup pre dopytovanie prebieha v prirodzenom jazyku. V sfére moderných informačných systémov predstavuje veľmi populárny (často samozrejmý) spôsob interakcie s webovou stránkou, aplikáciou. [1] Fundamentálnymi vlastnosťami fulltextového vyhľadávania sú:

- relevantnosť a skóre vrátených výsledkov
- rýchlosť vrátenia výsledkov
- transformácie vstupu typické pre prirodzený jazyk (základ slova, množné číslo, synonymá ...)

Fulltextové hľadanie aplikuje vstup na dátovú doménu a pomocou algoritmov typických v tejto oblasti (popísané neskôr) pridelí každému výsledku skóre, ktoré predstavuje relevanciu. Výsledky sú typicky zoradené na základe skóre a používateľ má možnosť vybrať si najrelevantnejší výsledok, prípadne modifikovať svoj vstup. Táto interakcia je z pohľadu používateľa veľmi priamočiara a rýchla, avšak kvalitné riešenie fulltextového hľadania môže v pozadí predstavovať zložitú logiku. [1] [2]



Obr. 1: Ukážka fulltext vyhľadávania na webe CZC.cz

2.1 Analýza súčasných riešení

Fulltextová funkcionalita v prostredí komplexných informačných systémov predstavuje veľmi efektívny nástroj a spôsob interakcie. Dopyt po kvalitných riešeniach sa prejavuje neustálym, kontinuálnym vývojom v tejto oblasti. V kapitole budú popísané vybrané fulltextové nástroje a bude zvolené riešenie, ktoré bude následne implementované.

2.1.1 Relačná databáza a fulltext

Uloženie dát v relačnej schéme predstavuje základný koncept perzistencie informačného systému. Získanie dát na základe vstupu je priamočiare a odpovedá jednoznačne zostavenému SQL dopytu. Relačné databázy implementujú mechanizmy pre hľadanie vzoru vstupného textu nad zvoleným stĺpcom, resp. množinou dát, jedná sa o :

- operátor LIKE, prípadne použitie regulárneho výrazu
- fulltextový index

Oba mechanizmi sú dnes podporované vo všetkých používaných riešeniach pre relačné databázy, ako MySQL, SQL Server, Oracle a pod. Kým výstup operátora LIKE je binárny (vzor sa zhoduje a záznam je vrátený, alebo nie), použitie fulltextového indexu operuje s vypočítaným skóre. Spomenuté mechanizmy majú svoje uplatnenie, avšak pri reálnom použití pozorujeme nevýhody :

- rýchlosť - relačná schéma sa typicky vyznačuje riadkovým uložením dát a práve tento fakt predstavuje pre fulltextové hľadanie základný problém. Riešenia ktorá sa orientujú výhradne na fulltextovú funkcionalitu ukladajú dáta v inom formáte, typicky sa jedná o bezschémové, príp. stĺpcové uloženie.

- vyťaženie relačnej databázy - relačná schéma informačného systému plní súčasne mnoho úloh: selektovanie, vkladanie, aktualizácia dát, réžia pre vytvorené indexy a pod. Fulltextový dopyt predstavuje ďalšiu operáciu, ktorá je v súvislosti s riadkovým uložením výpočtovo náročná. Výsledkom je vysoké vyťaženie databázy, čo môže spôsobovať predĺženie vykonávania operácií a zníženie priepustnosti. Relačné databázy nie sú navrhnuté ako distribuované úložisko, škálovanie výkonu je možné vykonávať optimalizáciou a horizontálnym škálovaním konkrétneho servera. Spojením všetkých faktorov sa z databáze môže stať úzke hrdlo, ktoré ovplyvňuje výkon a funkčnosť celého IS.
- kvalita – definícia fulltextového indexu poskytuje len obmedzené možnosti. V reálnom informačnom systéme je relevancia často ovplyvnená kalkulovaním s ďalšími faktormi, ako napríklad obľúbenosť vyhľadávaného výrazu, alebo iné formy určovania priority. Fulltextové indexy relačných schém natívne neumožňujú takýto prístup ovplyvňovania skóre.

Z dôvodov uvedených v tejto časti nebude fulltextová funkcionalita implementovaná v rámci relačnej schémy, resp. pomocou relačnej databázy.[3]

2.1.2 Apache Lucene

Apache Lucene (označované aj Apache Lucene Core) je knižnica kompletne napísaná v jazyku Java. Ide o open-source projekt, ktorý pojednáva funkcionalitu fulltextového hľadania. Hlavnými cieľmi je poskytovanie vysokého výkonu, škálovateľnosti, platformová nezávislosť a široká škála dostupnej funkcionality v súvislosti s fulltext hľadaním, indexovaním atď.[7]

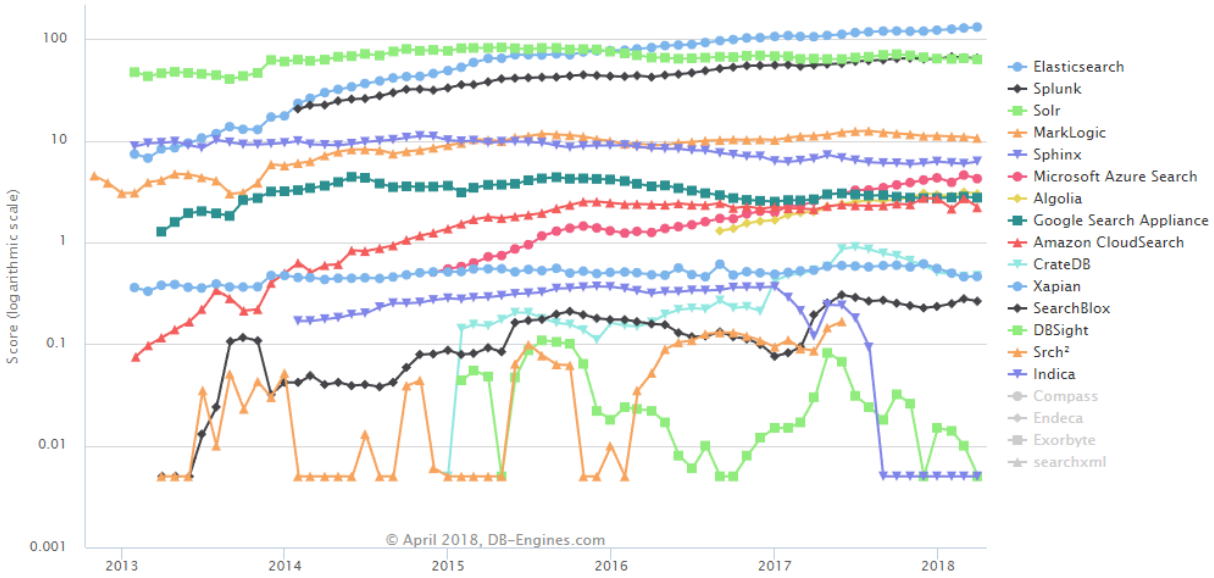
2.1.3 Apache Solr

Apache Solr je open-source framework, platforma pre účely vyhľadávania (fulltextového, parametrického) postavený nad knižnicou Apache Lucene. Apache Solr sa často označuje ako obal (wrapper) nad Apache Lucene z dôvodu, že poskytuje hotové riešenia ktoré možno integrovať. Pridáva funkcionalitu replikácie, cachovania, hľadania nad geografickými dátami, ale aj pohodlnejšie možnosti administrácie a API pre prístup k požadovaným celkom. Používanie výhradne Apache Lucene môže byť najmä pre počiatky implementácie náročné, keďže používanie a implementácia je výhradne v doméne vývojára. Lucene sa typicky používa pre integráciu do desktopových aplikácií, alebo v prípade nutnosti implementácie veľmi špecifickej funkcionality.[8]

2.1.4 Elasticsearch

Elasticsearch je vyhľadávací mechanizmus založený na technológií Lucene. Poskytuje fulltextové, parametrické vyhľadávanie, web rozhranie a podporuje bezschémové uloženie JSON dokumentov. Jedná sa o open source softvér. Elasticsearch vo svojom jadre používa Lucene indexy. Technológia je z tohto pohľadu porovnateľná s Apache Lucene. Elasticsearch prináša kvalitnejší model distribúcie a využívania techniky replikovania. Je preto výkonnejší a vhodnejší pre vyhľadávanie

v reálnom čase, čo predstavuje jeho hlavnú výhodu. Elasticsearch sa stal veľmi populárnym riešením, čo dokazuje značná miera dostupnej funkcionality pre širokú škálu vyhľadávacích scenárov. Dokumentácia je priamo prepojená s analytickými nástrojmi pre testovanie študovaných príkladov, doplnená o články a blogy, ktoré detailnejšie popisujú problematiku. Pre implementáciu fulltextového hľadania bolo zvolené riešenie Elasticsearch.



Obr. 2: Trendy vyhľadávacích technológií podľa webu db-engines.com

Cluster, index a typ, shard a replika, dokument

S funkcionalitou služby Elasticsearch súvisí niekoľko fundamentálnych pojmov. Elasticsearch je NRT (Near Real Time) platformou, čo znamená, že existuje veľmi krátka latencia od publikácie a zaindexovania (integrácie do Elastic štruktúr, analyzátorov atď.) po dobu kedy je dokument vyhľadateľný (resp. viditeľný). Typicky táto doba trvá maximálne jednotky sekúnd.

Cluster je zoskupenie uzlov (node, resp. serverov), ktorý obsahuje všetky dáta a poskytuje centralizované indexovacie a vyhľadávacie funkcie naprieč všetkými uzlami. Samotným uzlom rozumieme server, ktorý sa podieľa na indexovaní a vyhľadávaní. Uzol (node) môže byť zaradený do ľubovoľného clustra definovaním menných konvencií. Správna definícia názvov clustrov a nodov je dôležitá pre korektné spojenie v rámci clustra a poskytovaní obsahu.

Dátovú jednotku, ktorá môže byť indexovaná pre účely fulltextového vyhľadávania nazývame dokument. V prostredí platformy Gloffer to predstavuje dátový celok o produkte, prípadne inej entite. Dokument je vo formáte JSON a v rámci Elasticsearch musí byť zaradený do určitého typu v rámci Indexu.

Index je kolekcia dokumentov podobnej (rovnakej) charakteristiky. Typickým príkladom je index zákazníckych dát, index produktových dát atď. Identifikácia indexu je taktiež prostredníctvom mena. V rámci daného indexu môžu existovať typy. Typ je logická kategorizácia v rámci

indexu. Typicky pre dokumenty, ktoré majú spoločné dátové polia je definovaný spoločný typ. Index i typ teda logicky štruktúrujú dáta, ich použitie môže byť v určitých prípadoch ekvivalentné. V rámci platformy Gloffer dáta zaradzujeme do jedného Indexu, ktorý ich logicky rozčleňuje podľa daného Typu. Existujú tu typy ako napríklad produkt, zákazník atď.

V indexe môže byť uložené veľké množstvo dát, ktoré môžu prevyšovať hardvérové kapacity daného stroja (uzlu), či spomaľovať požiadavky a odpovede z daného uzlu. Pre tieto účely existujú shardy, ktoré rozdeľujú index na menšie celky. Každý shard tak následne chápeme ako samostatný index, hosťovaný v ľubovoľnom uzle v rámci clustra. Shardy horizontálne rozdeľujú objem obsahu a poskytujú možnosť distribúcie a paralelizácie operácií. Repliky predstavujú mechanizmus pre ochranu dát a ich zálohu v prípade zlyhania shardu, alebo nodu. Repliky definujú kópie shardov v rámci indexu, ktoré dokážu obnoviť dáta, resp. zotaviť zlyhanú časť. Taktiež dokážu škálovať výkon v prípade potreby, keďže sú chápané ako záložné shardy. Táto funkcionalita je plne manažovaná technológiou Elasticsearch. [9][10]

2.2 Špecifikácia funkcionality

V tejto časti práce sa pojednáva definícia dátovej domény relevantnej pre fulltextové vyhľadávanie a samotná funkcionalita vyhľadávania.

2.2.1 Dátová doména

Vyvíjaný IS predstavuje produktový agregátor s ďalšou špecifickou funkcionalitou pre dopyt a ponuku. Pre tento typ softvéru sú typické katalógové produktové dáta a feedové produktové dáta. Katalógové dáta sú deduplikovaná množina produktov s agregovanými cenami (minimálna a maximálna cena) a ponukami (firma, ktorá daný produkt ponúka). Feedové produkty sú dáta, ktoré predajcovia manažujú a zverejňujú za účelom zvýšenia publikácie obsahu ich e-shopu, typicky práve v rámci produktových agregátorov. Tie vykonávajú proces párovania, mapujú produktové dáta z feedov na katalóg a agregujú požadované vlasnosti. Párovanie typicky pracuje so špecifickým tvarom názvu produktu, jeho zaradenia v rámci kategórie a ďalších atribútov ako EAN, ISBN a pod. Úspešnosť procesu párovania závisí od kvality a úplnosti katalógovej databázy, kvality feedu (dodržania pravidiel pre tvorbu názvu atď.) a samotného párovacieho algoritmu. Výsledkom je množina napárovaných produktov, ktoré sú agregované v rámci katalógu a množina nenapárovaných produktov. V informačnom systéme bude aplikovaná funkcionalita fulltextového hľadania na obe množiny. Všetky dáta pojednávané v tejto kapitole predstavujú dáta v češtine, vývoj je zameraný na fulltext v českom jazyku. Produktové informácie typicky pozostávajú z :

- názvu produktu – reťazec o dĺžke 5 až 100 znakov (Samsung Galaxy S8)
- popisu produktu – reťazec o dĺžke 0 až 250 znakov (Nový smartphone ...)

- vlastnostiach produktu (pre katalógový produkt)– súbor typu kľúč: hodnota (Fotoaparát: 8 megapixel)
- značke produktu – kľúčové slová (Samsung)
- zaradenia v rámci kategórie – konečná kategória, resp. strom kategórií (Elektronika - smartphone)

Pre fulltextový vstup v doméne produktových informácií je najrelevantnejší názov produktu, prípadne značka – tá sa však často vyskytuje v samotnom názve. Vlastnosti a popis bežne neobsahujú informácie, ktoré sú použiteľné pre fulltextové hľadanie.

Okrem informácií, ktorými je samotný produkt popísaný, je možné v IS evidovať ďalšie dynamické údaje ako napríklad počet nákupov produktu, obľúbenosť produktu v podobe kladných alebo záporných hodnotení a pod. Pre informačný systém, ktorý sa zaoberá produktovými agregáciami sú typické trendové impulzy v zmysle nového (modelu) produktu, výrazné zlacnenie predchádzajúcich modelov a pod. Pre fulltextové hľadanie však vstupom zostáva reťazec textu a preto v prípade, že chceme zabezpečiť, aby sa produkt vyskytoval v popredných výsledkoch vyhľadávania (práve kvôli vývoju trendov na trhu), musíme evidovať a spravovať ďalšie faktory, ktorými dokážeme výsledné skóre ovplyvniť. Môže sa jednať o atribúty predstavujúce prioritu produktu, prípadne kategórie.

- priorita cieľovej kategórie
- priorita produktu
- počet kladných hodnotení

Ďalšími dátovými doménami, pre ktoré bude implementované fulltextové vyhľadávanie sú kategórie produktov a značky.

2.2.2 Funkcionalita

Pre moderný prístup k fulltextovému vyhľadávaniu sú typické funkcionality :

- dopĺňania a navrhovania (completion and suggestion)
- plnohodnotné fulltext hľadanie

Dopĺňanie a navrhovanie fráz predstavuje rýchlu odozvu systému na aktuálne zadávaný vstup. Výsledok je v takmer reálnom čase vracaný používateľovi v podobe niekoľkých najrelevantnejších výsledkov. Táto interakcia prebieha asynchrónne v zmysle prekreslia stránky (resp. vykonania presmerovania). Hľadanie pomocou práve zadávaného vstupu sa typicky vykonáva pomocou zhody prefixu, resp. pre zaručenie rýchlosti odozvy toto hľadanie predstavuje z pohľadu komplexnosti jednoduchú variantu. Plnohodnotné vyhľadávanie môže trvať rádovo o jednotky sekúnd dlhšie, algoritmus môže aplikovať zložité metriky a priority a typicky prebieha presmerovanie, resp. prekreslenie stránky s výsledkami. Vo vyvíjanom IS budú implementované obe varianty vyhľadávania.

2.2.3 Odolnosť, distribuovanosť a dostupnosť (HA)

Pre každý informačný systém je dôležitá úroveň odolnosti voči výpadku servera alebo služby. Kým pre relačné databázy je problematika distribuovanosti komplikovaná, z pohľadu fulltextových nástrojov, ktoré sú typicky implementované pomocou NoSQL databáz je to jedna z fundamentálnych vlastností.

Distribuovanosť vrstvy sa prejavuje nie len v zmysle odolnosti, ale taktiež rozkladaní záťaže a tým pádom celkovej rýchlosti. V IS bude implementovaná funkcionalita, ktorá zabezpečí funkčnosť v prípade výpadku niektorého zo serverov zabezpečujúcich fulltextové hľadanie.

Dátová doména pre fulltextové hľadanie predstavuje vzhľadom na katalóg a proces párovania dynamickú množinu. V reálnom scenári sa dáta niekoľkokrát denne aktualizujú. Vyvinutá fulltextová funkcionalita musí byť schopná zabezpečiť aktualizácie dát počas poskytovania svojej funkcionality, resp. bude minimalizovaná nedostupnosť služby počas procesov aktualizácie a migrácie dát.[11]

2.3 Návrh

Návrh fulltextovej funkcionality spočíva v definovaní základnej štruktúry clustra. Štruktúra clustra sa skladá z indexov, v rámci ktorých sú definované typy objektov a samotné dátové polia. Nasledujúca tabuľka stručne zhŕňa návrh vytvorených indexov. Pre produkty sú podľa definície dátovej domény navrhnuté katalógové a feedové indexy. Návrh taktiež obsahuje indexy pre ďalšie entity ako kategória a značka.

V tabuľke 1 možno pozorovať oddelenie fulltextových indexov a indexov pre dopĺňanie. Toto rozdelenie je z dôvodu jednoznačného definovania indexov pre plnohodnotné a komplexné vyhľadávanie (prehľadávanie viac dátových polí, zložitejšia skórovacia funkcia) a indexov, ktorých primárnym cieľom je NRT (takmer reálny čas) získanie výsledkov (jednoduchšia skórovacia funkcia, prehľadávanie jedného dátového poľa).

Názov indexu	Typ indexu	Typ objektu	Fulltext dátové polia	Skóre faktory	Účel
glofer_data	F	pro_catalog	krátky názov, dlhý názov, popis		párovanie
		cat_category	názov	PK	fulltext hľadanie kategórií
		cat_trademark	názov	PT	fulltext hľadanie značiek
		cms_article	názov		administratíva
catalog	F	pro_catalog	krátky názov, dlhý názov, popis	PK, PP	fulltext hľadanie katalóg. produktov
catalog_completion	C	pro_catalog	krátky názov	PP	dopĺňanie, navrhovanie katalóg.produktov
feed	F	feed_product	názov		fulltext hľadanie nespár. produktov
feed_completion	C	feed_product	názov		dopĺňanie, navrhovanie nespár. produktov
category_completion	C	cat_category	názov	PK	dopĺňanie, navrhovanie kategórií
trademark_completion	C	cat_trademark	názov	PT	dopĺňanie, navrhovanie značiek

Tabuľka 1: Návrh vytvorenia fulltextových indexov

F – fulltext index, plnohodnotné fulltext hľadanie
C – completion index, dopĺňanie a navrhovanie
PK – priorita kategórie, manažované v rámci IS
PP – priorita produktu, manažované v rámci IS
PT – priorita značky, , manažované v rámci IS

Tabuľka 2 znázorňuje dátový objem jednotlivých indexov a intervaly aktualizácie dát.

Pre odolnosť voči výpadku nie je možné uviesť všeobecný návrh, keďže riešenie je priamo závislé na implementácií a z definície IS nie sú určené odpovedajúce číselné metriky.

Názov indexu	Počet záznamov	Aktualizácia dát
gloffer_data	10 - 70 milíonov	1x mesačne
catalog	10 000 – 1000 000*	1-3x denne
catalog_completion	10 000 – 1000 000*	1-3x denne
feed	10 000 – 1000 000*	1-3x denne
feed_completion	10 000 – 1000 000*	1-3x denne
category_completion	10 - 1000	1-3x denne
trademark_completion	10 - 1000	1-3x denne

* - tento údaj je priamo závislý na množstve feedových informácií (kvantita business domény IS) a kvalite párovacieho mechanizmu

Tabuľka 2: Kvantita záznamov fulltextových indexov

2.4 Implementácia

V časti bude popísaná implementácia clustra vzhľadom na požiadavky a navrhnuté indexy popísané v predošlých sekciách. Fulltextová funkcionalita bude implementovaná pomocou technológie Elasticsearch, konkrétne vo verzii 5.6.

2.4.1 Schéma clustra

Implementácia clustra spočíva v inštalácií a konfigurácii skupiny uzlov. Cluster môže byť v najjednoduchšej variante implementovaný jedným uzlom, ktorý plní všetkú funkcionalitu a obsahuje všetky dáta. Robustnosť takéhoto riešenia sa z pohľadu odolnosti a dostupnosti opiera o funkčnosť práve tohoto uzla a v prípade výpadku je nedostupná celá služba. Z časti špecifikácia zadania je dané, že implementovaná služba musí spĺňať určitú úroveň odolnosti a dostupnosti. V rámci technológie Elasticsearch sú pre tieto účely dostupné rôzne typy uzlov ako aj mechanizmus replikácie dát.

Master-eligible uzol

Plní funkciu riadiacej jednotky clustra. Zodpovedá za vytváranie a mazanie indexov, sleduje uzly zapojené v clustry, alokuje shardy a pod. Môže byť zvolený za hlavný uzol. Tento uzol môže obsahovať dáta, avšak v prípade tejto implementácie striktne odlišujeme riadiace a dátové uzly. Stabilný master uzol je podstatný pre funkčnosť a stabilitu celého clustra.[10]

Data uzol

Obsahuje dáta zaradené do indexov a typov. Tento uzol plní funkcionalitu indexovania (vkladania nového obsahu, aktualizácie atď.) a taktiež dopytovania (vyhľadávanie, agregovanie atď.) Všeobecne plní CRUD dátové požiadavky. Počet dátových uzlov v clustry odpovedá dátovému

objemu. V prípade nízkeho počtu dátových uzlov a objemnej dátovej domény môže nastať zahltenie a preťaženie uzlov.[10]

Ingest uzol

Uzol dokáže transformovať dokument (zmeniť obsah, pridať dodatočné informácie atď.) pri procese indexovania. Použitie tohto dedikovaného uzla sa odporúča v prípade veľkého množstva dokumentov, ktoré chceme pred indexovaním upravovať.[10]

Tribe uzol

Špeciálny typ koordinačného uzla, ktorý sa dokáže pripojiť k viacerým clustrom a vykonávať funkcionalitu naprieč nimi. Typy uzlov je možné skladať, resp. vytvoriť jeden uzol, ktorý zastáva funkcie master, data, ingest, ale taktiež explicitne rozdeliť kompetencie.[10]

Problém split brain

Pomocou vyššie uvedených typov je možné zostrojiť cluster obsahujúci riadiace, transformačné a dátové uzly. Distribuované systémy ako Elasticsearch monitorujú svoje uzly a v prípade výpadku prebieha niekoľko operácií, ktorých cieľom je zachovanie stability a dostupnosti. Typicky je jeden uzol označený ako master a predstavuje mozog celého clustra. V prípade výpadku je prvou a hlavnou operáciou voľba (nového) master uzla. Táto operácia prebieha interne a za master uzol je zvolený niektorý z master-eligible uzlov vzhľadom na stav clustra, ostatých uzlov. Po voľbe nového master uzla môžu prebiehať migrácie dát z dôvodu rozloženia záťaže, replikácia dát atď.

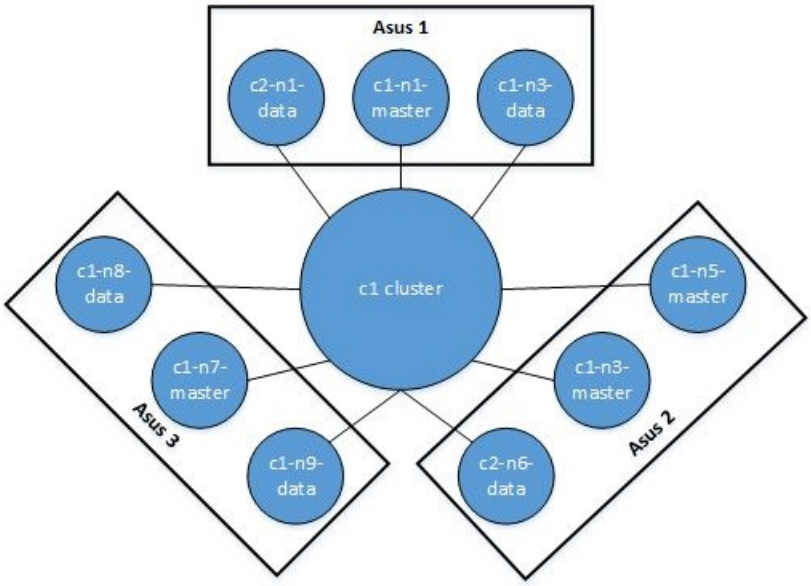
Problém split brain pojednáva práve situácie výpadku, kedy nastáva voľba master uzla a znovu - zostrojenie clustra. Pre tieto scenáre je podstatná konfiguračná anotácia discovery. Jej cieľom je zabránenie straty dát a nedostupnosti služby. Táto konfigurácia je definovaná v každom master-eligible uzle a udáva minimálny počet dostupných (viditeľných) master-eligible uzlov, ktoré sú nutné k sformovaniu clustra. Táto hodnota je defaultne nastavená na 1, problematika bude následne vysvetlená na príklade.

Pojednávajme cluster zložený z 2 master-eligible uzlov (môžu taktiež obsahovať dáta) a hodnotu discovery.zen.minimum_master_nodes nastavenú na 1. V prípade sieťového (alebo iného) výpadku medzi danými dvoma master-eligible uzlami nastáva situácia, že každý master-eligible uzol vidí len sám seba. Hodnota minimum_master_nodes = 1 umožňuje korektné sformovanie clustra a oba master-eligible uzly sa považujú a zvolia za nový master uzol. Výsledkom sú dva clustre, alebo problém nazývaný split brain. Spomenuté uzly sa nespoja do spoločného clustra, kým sa jeden z uzlov nereštartuje. Všetky dáta, ktoré sa medzi časom mohli zapísať do uzla, ktorý je nutné neskôr reštartovať budú stratené. Ďalší scenár pojednáva cluster z 3 master-eligible uzlov a hodnotu discovery.zen.minimum_master_nodes = 2. Ak výpadok oddelí niektorý z master-eligible uzlov, strana s jedným master-eligible uzlom nevidí dostatočný počet ďalších master-eligible uzlov (2). Nie je tak možné zvoliť sa za nový master uzol a sformovať

cluster. Strana s dvoma master-eligible uzlami podmienku spĺňa, niektorý z uzlov sa označí za master a korektne sa sformuje nový cluster. Akonáhle je výpadok opravený, strana s oddeleným master uzlom sa zapojí do novo-existujúceho clustra a pokračuje vo svojej funkcii. Pre scenáre popísane nižšie bola definovaná formula, ktorá určuje korektné nastavenie konfigurácie discovery.zen.minimum_master_nodes :

$$\text{discovery.zen.minimum\_master\_nodes} = (\text{počet master-eligible uzlov} / 2) + 1$$

Nasledujúci obrázok znázorňuje schému clustra. Je celkovo vytvorených 9 uzlov – 3 master uzly a 6 dátových. Dané uzly sú vytvorené ako virtuálne servery a schéma taktiež znázorňuje ich existenciu v rámci fyzických serverov (Asus 1, Asus 2 a Asus 3). Pre danú konfiguráciu je discovery.zen.minimum_master_nodes nastavené na hodnotu 2. [9][10]



Obr. 3: Schéma fulltext clustra

2.4.2 Inštalácia a konfigurácia uzla

Uzly clustra sú vytvorené ako Elasticsearch služby bežiace na virtuálnych serveroch. Operačný systém virtuálnych serverových technológií je Ubuntu 16.04 (CLI). Požiadavky na inštaláciu :

- nainštalovaná Java verzie 8
- nakonfigurovaná sieťová infraštruktúra
- dostatok dostupného miesta

Samotná inštalácia prebieha pomocou stiahnutie tar archívu ktorý je vystavený v rámci repozitára Elasticsearch:

```
$ curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-5.6.8.tar.gz
```

Archív je následne rozbalený do cieľovej zložky :

```
$ tar -xvf elasticsearch-5.6.8.tar.gz
```

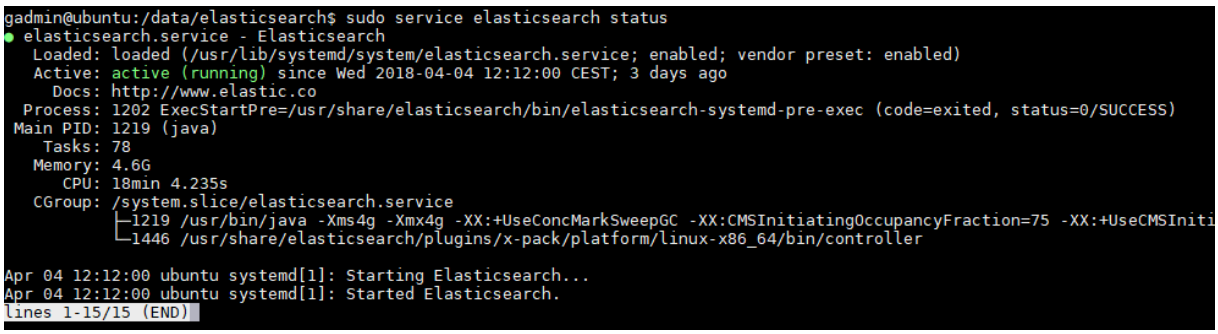
Pre spustenie služby je nutné spustiť skript, ktorý sa nachádza v zložke bin v rozbalenom archíve :

```
$ ./elasticsearch-5.6.8/bin/elasticsearch
```

Pre overenie môžeme skontrolovať stav služby pomocou príkazu :

```
$ sudo service elasticsearch status
```

a skontrolovať výstup :



Obr. 4: Spustená Elasticsearch služba

Popísaný proces predstavuje základnú inštaláciu služby. Pre zostrojenie clustra podľa definovaných kritérií je nutné daný uzol nakonfigurovať pomocou konfiguračného súboru elasticsearch.yml, ktorý sa vytvorí po inštalácií v rámci adresára /etc/elasticsearch.

Konfiguračný súbor poskytuje mnoho možností konfigurácie. Nasledujúca tabuľka zhŕňa potrebné nastavenia pre popisovanú implementáciu :

Konfigurácia	Hodnota	Popis
cluster.name	reťazec, napr. c1	názov clustra
node.name	reťazec, napr. c2	názov uzla
node.master	true / false	typ uzla : master
node.data	true / false	typ uzla : data
node.ingest	true / false	typ uzla : ingest
path.data	cesta, napr. /data/elasticearch	cesta k úložisku dát
path.logs	cesta, napr. /data/elasticearch/logs	cesta k log súborom
network.host	IP adresa	bind IP adresa
http.port	číslo, defaultne 9200	číslo portu pre komunikáciu
discovery.zen.ping.unicast.hosts	zoznam IP adries	IP adresy master uzlov
discovery.zen.minimum_master_nodes	číslo	minimálny počet viditeľnych master uzlov
path.repo	cesta, napr. /data/nfs	cesta k repozitáru pre ukladanie záloh

Tabuľka 3: Možnosti konfigurácie uzla

Implementácia služby sa skladá z master a dáta uzlov. Inštalácia služby Elasticsearch na uzly-servery prebieha pre oba typy rovnako. Z tabuľky vyššie je možné pozorovať, že odlíšenie daných typov prebieha len pomocou nastavení hodnôt pre konfigurácie node.master, node.data a node.ingest. Master uzol bude mať nastavenú hodnotu true pre konfiguráciu node.master a hodnotu false pre konfiguráciu node.data a node.ingest. Analogicky, dátový uzol bude mať nastavenú hodnotu true pre konfiguráciu node.data a hodnotu false pre konfiguráciu node.master a node.ingest. Zvyšné nastavenie môžu zostať nezmené za predpokladu, že virtuálne servery sú nakonfigurované rovnako.

2.4.3 Vytvorenie indexu

Vytvorenie indexu spočíva z dvoch častí – nastavenia samotného indexu, analyzátorov a definícia mapovania pre obsah. Nasledujúce konfigurácie prebiehajú podľa dokumentácie Elasticsearch, pomocou odpovedajúcich API. Manažment a nastavenia indexov (typov, mapovania, stav) sú dostupné v rámci Indicies API. Dané príkazy je možné volať ako cURL požiadavky zasielané na odpovedajúci uzol. Požiadavky je možné zasielať na ľubovoľný uzol (dáta, master, ingest), avšak v prípade príkazov pre manažment (resp. Indicies API, Cluster API) sa odporúča smerovať ich na master-eligible uzly. Obsah dopytov, ako aj dáta sú v technológií Elasticsearch zasielalané vo formáte typu JSON.

Nastavenia indexu
Táto časť obsahuje 3 základné konfigurácie :

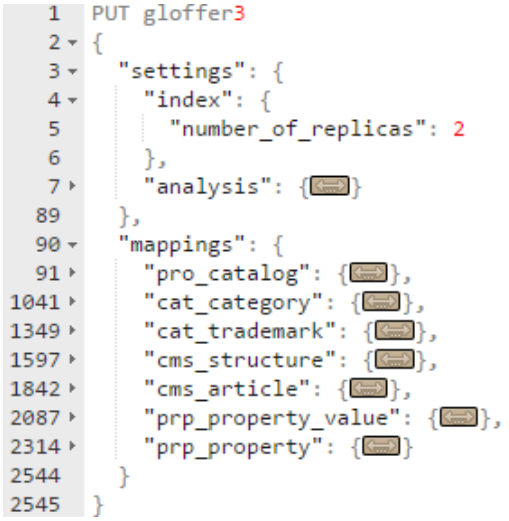
- number_of_shards – počet shardov (Pojmy: cluster, index a typ, shard a replika, dokument), prednastavená hodnota je 5
- number_of_replicas – počet replík (Pojmy: cluster, index a typ, shard a replika, dokument) prednastavená hodnota je 1
- analysis – pojednávané v nasledujúcom texte

Hodnoty počtu shardov a replík závisia na viacerých faktoroch – počtu uzlov, počtu indexov, dátovej náročnosti. Pri vytvorení indexu sa shardy a repliky distribuujú na ostatné uzly, čo predstavuje mechanizmus škálovania výkonu a HA v prípade výpadku. Dané počty sa špecifikujú pri vytváraní indexu a ich zmena vyžaduje reindexáciu všetkých dát v indexe. [9][10]

Jedným z bežných prístupov je definovanie počtu shardov podľa počtu uzlov. V konkrétnej implementácií uvažujeme o počte dátových uzlov, keďže master uzly nijaké dáta neobsahujú. Nastavenie spôsobí, že na každom (dátovom) uzle sa alokuje shard, čo predstavuje relatívne rovnomerné rozloženie záťaže. Repliky predstavujú záložné shardy v prípade výpadku, alebo nutnosti navýšenia výkonu, v bežnom stave sú neaktívne. Hodnota počtu replík = 1 znamená, že pre každý alokovaný shard sa alokuje jedna replika.

S nastavením indexu je nutné pojednávať systém ako celok – pri vytvorení príliš mnoho shardov nastáva preťaženie a mechanizmus distribúcie stráca význam. V prípade replík je dôležitým faktorom objem dát. Replikovanie môže spôsobiť skokový nárast veľkosti daného indexu.

V implementovanom riešení zostávajú prednastavené hodnoty – vzhľadom na počet uzlov a robustnosť riešenia sa jedná o adekvátnu konfiguráciu.



Obr. 5: Ukážka špecifikácie indexu

Definícia mapovania

Mapovaním rozumieme logický popis dátových polí v zmysle dátového typu, formátu a špeciálnych tagov pre účely indexovania a vyhľadávania. Elasticsearch poskytuje niekoľko prístupov ako mapovanie definovať. V základnej konfigurácií je nastavené tzv. dynamické mapovanie, ktorým sa automaticky označujú dátové polia príslušným dátovým typom. Dátový typ sa v rámci Elasticsearch definuje pre korektné fulltextové funkcie, resp. správny prístup k dátam pri ich spracovávaní. Táto voľba predstavuje najrýchlejší prístup, pretože bez nutnosti ďalšieho nastavenia sa môže prikročiť k publikácií dát a ich používaniu. Nevýhodou je, že Elasticsearch nemusí dáta zaradiť do správneho dátového typu. Zostrojený obsah môže byť chybný, alebo v nesprávnom formáte a dôsledkom môže byť znemožnená, alebo inak negatívne ovplyvnená funkcionalita indexovania a fulltextového vyhľadávania.

Ďalším prístupom je vypnutie dynamického mapovania, resp. nastavenie striktného režimu, ktorý je použitý aj v popisovanom riešení. Striktný režim spočíva v manuálnom nastavení mapovania - manuálna definícia dátových typov, formátu a špeciálnych tagov pre fulltextové vyhľadávanie. Takto definované mapovanie zabezpečuje, že dáta sú indexované v očakávanej podobe. Ak dátový balík (dokument JSON) obsahuje dátové pole, ktoré sa nezhoduje s definovaným mapovaním, takýto dokument nie je prijatý a indexovaný. Riešenie v rámci platformy Gloffer využíva striktný režim pre zaistenie potrebného formátu dát v každom prípade. Dynamické

mapovanie bolo po testovaní tejto varianty vypnuté, pretože dátové balíky v čase vývoja obsahovali chyby, ktoré sa následne propagovali pri publikácií do Elasticsearch. Výsledkom bola nepoužiteľná logika dopytovania dát, resp. hodnotenia relevancie výsledkov.

Nasledujúca tabuľka zhŕňa dostupné dátové typy. Pre mapovanie dátových polí boli použité základné, komplexné a geografické typy. Polia, ktoré majú byť dostupné pre fulltextové vyhľadávanie nesú príznak text, pre popisované riešenie sa takto označujú polia názvov, popisov, ich prekladov a podobne. Keyword označuje polia, ktoré sú používané pre vyhľadávanie v zmysle zhody dát, v riešení platformy Gloffer sú takto označené napríklad konštanty, ktoré sa používajú pre parametrické vyhľadávanie alebo agregácie. Numerické dátové polia sú používané pre identifikátory zo zdrojovej tabuľky a bežné numerické dáta. Použitie dátumových a boolean typov je implicitné. Komplexné dátové typy sa používajú na označenie skupiny dát, resp. predstavujú komplexnejší dátový celok. Typ Pole sa používa pre označenie kolekcie niekoľkých hodnôt, podmienkou ale je zhodný dátový typ týchto hodnôt. Typ Objekt je defaultný dátový typ, označuje hierarchickú skupinu dát s ľubovoľne definovanými dátovými typmi v rámci neho. Vnorená štruktúra (Nested) je špeciálnym prípadom typu Objekt, ktorá umožňuje poľu objekov byť indexované a dopytované nezávisle. Elasticsearch označenie Objekt konvertuje do plochej štruktúry, takže interne zaniká hierarchia aj nezávislosť dát. Vnorená štruktúra umožňuje zachovať nezávislosť tak, že každý podcelok indexuje a dopytuje samostatne.

Skupina	Typ	Dátový typ	Popis a použitie
Základné	Textové (string)	text	fulltextové pole
		keyword	štrukturovaný obsah, pre agregácie, usporiadanie atď.
	Numerické	long, integer, short, byte, double, float	číselné hodnoty, id hodnoty
	Dátumové	date	dátumové a časové hodnoty
	Boolean	boolean	Logické hodnoty 1 a 0
	Rozsahové	integer_range, float_range, long_range, double_range, date_range	rozsahy numerických hodnôt
Komplexné	Pole	výčet hodnôt rovnakého typu	
	Objekt	hierarchická štruktúra, rôzne dátové typy, defaultný typ	
	Vnorená štruktúra	špecializovaná verzia Objekt, poskytuje nezávislé indexovanie (závislé agregácie atď.)	
Geo dáta	Geo bod	dáta o zemepisnej šírka a výške	
	Geo tvar	špecializované geoinformácie, polygóny	
Špecializované	IP	špecializované tvary dát	
	Token		

Tabuľka 4: Popis dátových typov Elasticsearch [10]

Popísané nastavenia možno špecifikovať pre každý navrhnutý index. Implementácia definuje potrebné mapovania (a analyzátory) pre každý index zvlášť, avšak nastavenie počtu shardov, replík je rovnaké pre každý index.

Obr. 6: Označenie, mapovanie dátových polí typu pro_product

Skripty pre vytvorenie navrhnutých indexov, vrátane mapovania a analyzátorov sa nachádzajú v prílohách.

2.4.4 Analyzátory

Fulltextové vyhľadávanie v Elasticsearch prebieha nad dátovými poľami, ktoré sú typu text. U daného dátového poľa je možné definovať analyzátor, ktorý určuje spôsob indexovania (zaradenia do fulltextových štruktúr) a následného hľadania. Analyzéry zodpovedajú, akým spôsobom sú dáta vyhľadávané a priamo súvisia s relevantnosťou a skóre vrátených výsledkov. Tematika a možnosti jazykových analyzátorov v Elasticsearch sú veľmi komplexné. V tejto časti budú popísané základné prístupy a implementované riešenia, ktoré vzhľadom na testovanie dosahovali najlepšie výsledky. Testovanie analýzy textu a hľadanie ideálnej varianty analyzátora je dlhodobý proces, ktorý nie je možné automatizovať. Popísané prístupy sú výsledkom pokusov a testovania rôznych konfigurácií.

Podpora funkčnosti fulltextového vyhľadávania pre definované dátové pole začína v čase indexovania – vkladania dokumentu a pokračuje v čase dopytovania – vyhľadávania dokumentu. Elasticsearch poskytuje možnosť definovania rôznych analyzérov v čase indexovania a v čase vyhľadávania. Vo väčšine prípadov chceme k indexovaniu a hľadaniu pristupovať uniformným spôsobom a definujeme len jeden typ analyzátora. Existujú však scenáre, kedy je takýto prístup neefektívny a môže negatívne ovplyvniť, alebo znefunkčniť fulltextové hľadanie. Typickým príkladom sú nGramy. Ak indexujeme slovo „elasticsearch“ pomocou analyzátora typu nGram, s nastavením minimálneho počtu ngramov = 3 a maximálneho počtu ngramov = 20, v Elasticsearch štruktúre sa objaví nasledovná sekvencia:

„ela“, „elas“, „elast“, „elasti“, „elastic“, „elastics“, „elasticse“, „elasticsea“, „elasticsear“, „elasticsearc“, „elasticsearch“

V prípade dopytovania slova *„elasticsearch“* sme zchopný nájsť čiastočnú, alebo celkovú zhodu vzhľadom na indexovaný vstup a vrátiť výsledok. Nie je nutné (a v určiom zmysle môže byť nesprávne) aplikovať techniku nGramov aj na dopytovací vstup. Ak by bol dopytovací vstup *„elastic“* spracovaný formou nGramov, vyzeral by nasledovne :
„ela“, „elas“, „elast“, „elasti“, „elastic“

Takýto vstup je oveľa komplexnejší a mohol by viesť k mätúcim výsledkom. V prípade, ak by sme indexovali slovo *„elaborate“* postupom popísaným vyššie a používali nGram analyzátor aj na vyhľadávanie, nGram *„ela“* by spôsoboval zhodu oboch dokumentov pre dopyt na slovo *„elastic“*. [9][10]

V konkrétnej implementácií sa testuje predovšetkým spôsob uniformného prístupu analyzátorov k indexovaniu a hľadaniu, resp. používa sa jeden analyzátor. nGramy sa používajú predovšetkým pre hľadanie zhody na základe prefixu, reálne tak dokážu implementovať funkcionalitu dopĺňania a navrhovania na základe prefixového vstupu. Táto funkcionalita môže byť implementovaná na základe vlastných požiadaviek, napríklad používaním spomenutého nGramu, avšak pre tieto účely Elasticsearch definuje dátové pole typu Completion, ktoré interne vyhľadáva prefixovo. Tento typ dátového poľa v rámci indexu je optimalizovaný pre rýchlosť (dopĺňanie a navrhovanie podľa toho ako používateľ zadáva vstup vyžaduje čo najrýchlejšiu odozvu), je uložený in-memory.

V časti Návrh boli definované indexy, ktoré sú označené ako Completion – tieto indexy implementujú fulltextové hľadanie na základe prefixového vstupu práve pomocou vstavaného dátového poľa typu Completion.

V rámci Elasticsearch existujú vstavané a vlastné analyzátory. Analyzátor predstavuje súbor, ktorý obsahuje 3 stavebné bloky :

- znakový filter – pridáva, odstraňuje a upravuje vstupný stream na úrovni znakov, je možné ho používať na odstránenie HTML tagov. Analyzér môže obsahovať 0 a viac znakových filtrov.
- tokenizer – vstupný stream rozloží na individuálne tokeny (väčšinou na úrovni slov) na základe separátora (whitespace znak, slovo). Analyzátor musí mať definovaný práve 1 tokenizer.
- token filter – prijíma vstupný stream ako tokeny (výstup tokenizera), pridáva, odstraňuje a upravuje tokeny. Analyzér môže obsahovať 0 a viac token filtrov, sú aplikované v definovanom poradí. Patria tu napríklad filtre, ktoré prevádzajú tokeny na malé písmená, odstraňujú stop slova (predložky atď.), pridávajú synonymá a pod.

Vstavané analyzátory dané stavebné bloky implementujú vlastným, interným spôsobom a vstavaný analyzátor sa používa ako celok. Vlastným analyzátorom môžeme definovať a konfigurovať každý stavebný blok podľa vlastných požiadaviek a určovať vlastné poradie vykonania

filtrov. V rámci implementácie fulltextovej funkcionality bol testovaný vstavaný Elasticsearch analyzátor pre češtinu a niekoľko variant vlastných analyzátorov.

Analýza a priebežné testovanie

Testovanie jazykových analyzátorov a ich možností prebiehalo v dvoch fázach. Prvou fázou je analýza pomocou Analyze API. V rámci Analyze API je možné definovať stavebné bloky analyzátora, zadať vstup a sledovať transformácie textu. Analyze API je vhodné k pochopeniu aké transformácie a úpravy textu prebiehajú. Druhou fázou je vytvorenie konkrétneho indexu s definovaním analyzátorov, vložením testovacích dát a testovaním vyhľadávania ako celku.

Obr. 7: Analýza spracovania textu pomocou Analyze API

Tokenizéry

Tokenizéry určujú, aké tokeny budú poslané v zreťazenom spracovaní do token filtrov. Existuje niekoľko možností, ako vstupnú sekvenciu slov rozdeliť na jednotlivé tokeny.

Dôležitým poznatkom zreťazeného spracovania, ktoré sa skladá z tokenizéra a token filtrov je, že môžeme dosiahnuť podobnú (v určitých scenároch rovnakú) funkcionalitu používaním rôznych kombinácií tokenizérov a filtrov. Príkladom je použitie Keyword tokenizéru spolu s Word Delimiter token filtrom, verzus použitie Standard tokenizéra, alebo Whitespace tokenizéra bez token filtra pre delenie. Keyword tokenizér funguje spôsobom, že text nedelí, ale vracia jeden token ktorý predstavuje vstupnú sekvenciu a rozdelenie na tokeny vykonáva v tomto prípade Word Delimiter token filter. Standard alebo Whitespace tokenizéry vykonajú delenie textu na tokeny na úrovni tokenizéru, nie je tak nutné definovať token filter, ako napríklad Word Delimiter.Elasticsearch definuje niekoľko tokenizérov :

- tokenizéry orientované na slová – zvyčajne delia text na slová, patrí tu Standard, Letter, Whitespace, Keyword atď., v texte budú popísané tokenizéry relevatntné pre implementáciu
- tokenizéry orientované na čiastočné, neúplné slová – patrí tu N-Gram a Edge N-Gram tokenizer, dané tokenizéry nie sú v implementácií pojednávané z dôvodu použitia špecializovaného dátového typu Completion a odpovedajúceho indexu pre vyhľadávanie pomocou čiastočného vstupu
- tokenizéry pre štrukturovaný text – spracovanie textu v podobe email adries, identifikátorov a podobných špecializovaných štrukturovaných vstupov

Fulltextové hľadanie tejto implementácie je orientované na hľadanie zhody celých slov. Relevantné tokenizéry v tejto oblasti sú tokenizéry :

- Standard – delí text na tokeny, delenie je definované na úrovni hraníc slov podľa algoritmu Unicode Text Segmentation
- Letter – delí text na tokeny na základe znakov, ktoré nie sú písmenom; delenie prebieha po narazení na číslo, medzeru, pomlčku, apostrof atď.
- Whitespace – delenie prebieha na úrovni akéhokoľvek whitespace znaku, ako naprílad medzera, tabulátor atď.
- Keyword – špeciálny typ, nedelí, len vracia text ako token a delenie prenecháva ďalším mechanizmom

Token filtre

V časti budú popísané token filtre pre úpravu tokenov na úrovni jazyka a filtre pre všeobecné transformácie textu. Token filtre sa zreťazene vykonávajú v definovanom poradí.

Synonymá a stop slová

Stop slovom rozumieme pojem, ktorý sám o sebe nenesie žiaden význam. Ide predovšetkým o spojky, predložky. Používanie synoným a stop slov možno definovať v token filtry.

Synonymá pre český jazyk nie sú súčasťou Elasticsearch. Ich integrácia spočíva v stiahnutí, alebo vytvorení súborov v odpovedajúcom tvare. Elasticsearch podporuje formát Solr a WordNet. Formát Solr sa bežne používa v platforme OpenOffice. Odpovedajúce súbory je možné stiahnuť zo stránok http://www.openoffice.org/ a integrovať-nakopírovať do konfiguračných zložiek všetkých uzlov. Konfiguračná zložka sa v prípade inštalácie na Unixový systém nachádza v /etc/elasticsearch. Stop slová pre český jazyk sú integrované v Elasticsearch, inštalácia vlastných zdrojov je analogická ako pre synonymá. Po integrovaní súborov na všetky uzly a reštartovaní služby je možné používať synonymá v definovaní analyzéru v rámci indexu.[10]

```
{
  "settings": {
    "index": {
      "number_of_replicas": 1,
      "number_of_shards": 5
    },
    "analysis": {
      "filter": {
        "synonyms_CZ": {
          "type": "synonym",
          "synonyms_path": "syn_cs.txt",
          "tokenizer": "keyword"
        },
        "stopwords_CZ": {
          "ignore_case": "true",
          "type": "stop",
          "stopwords": [
            "_czech_"
          ]
        }
      }
    }
  }
}
```

Obr. 8: Ukážka definície synoným a stop slov

Dátová doména fulltextového hľadania zahŕňa niekoľko variant produktového názvu, názvu kategórie a značky (v českom jazyku). Testovaním analyzátorov pre vstupy typické v tejto business doméne sa dospelo k záveru, že používanie filtrovania stop slov je zbytočné, keďže vstupy typicky obsahujú dané názvy a kľúčové slová bez stop slov. Reálne používateľ v produktovej doméne hľadá názov alebo fragment názvu produktu, resp. vstup nie je definovaný formou súvislých viet, ktoré bežne stop slová obsahujú.

Synonymá môžu výrazne zlepšiť možnosti hľadania, avšak synonymá, ktoré sú dostupné v rámci OpenOffice obsahujú len minimum synoným použiteľných v produktovej dátovej doméne. OpenOffice synonymický slovník obsahuje množstvo synoným pre typické slová, avšak vzhľadom na produktovú doménu neprináša zvýšenie relevantnosti. Testovanie a fakty popísané v tejto časti sú dôvodom nepoužívania synoným, resp. nepoužívanie synonymického slovníka OpenOffice. Riešením je vytvorenie vlastného synonymického slovníka pre konkrétnu doménu dát, tento krok však nie je predmetom implementácie.

Stemmer pre český jazyk

V rámci implementácie analyzéru pre produktovú doménu je možné využiť vstavané Elasticsearch stavebné bloky pre český jazyk. Jedná sa o vstavané stop slová a vstavaný stemmer českého jazyka. Vylúčenie stop slov z analýzy bolo vysvetlené v časti vyššie. Stemmerom rozumieme algoritmus, ktorý dokáže previesť slovo na jeho základný tvar. Pre vstupné slovo-token „mobilný“ je výstupom, resp. ďalším tokenom slovo „mobil“. Stemmer v Elasticsearch predstavuje token filter. Pre fulltextové hľadanie v rámci produktovej domény môže tento prístup zlepšiť relevantnosť, je dohľadateľná zhoda medzi vstupom „mobilný telefón samsung“ a inde-

xovanými dátami v tvare „mobil samsung S8“´(názov produktu), „mobil“ (názov kategórie) a pod. Stemmovanie môže byť implementované dvoma spôsobmi :

- algoritmický stemmer – aplikuje pravidlá transformácie na základný tvar pre každé slovo
- slovníkový stemmer – aplikuje stemmovanie slov podľa slovníka pre daný jazyk

Oba prístupy dokážu poskytovať kvalitné výsledky, avšak predstavujú značný rozdiel v prístupe. V praxi a pokročilých stemmeroch sa dané prístupy kombinujú. Algoritmický stemmer môže aplikovať pravidlá, ktorých výsledkom môže byť nesprávne slovo, resp. zlé odvodenie základného tvaru. Pravidlá pre odvodenie základu slova sa aplikujú na každý token rovnakým spôsobom. Kvalita riešenia spočíva v definovaní správnych jazykových pravidiel. Kvalita slovníkového stemmera odpovedá kvalite slovníka. Ak dané slovo nie je v slovníku, nie je možné určiť jeho tvar (algoritmický stemmer by aplikoval pravidlá). Slovníkové stemmery môžu obsahovať mnoho slov, prefixov a sufixov. Z dôvodu výkonu a snahe o výsledky v reálnom čase sú dané dáta načítané v operačnej pamäti, čo predstavuje výkonnostný faktor. Všeobecne sú algoritmické stemmery považované za rýchlejšie oproti slovníkovým.[10]

V implementovanom riešení sa používa slovníkový stemmer z dôvodu, že dané slovníky sú pre český jazyk integrované v Elasticsearch, ale sú taktiež dostupné externé slovníky. Algoritmický stemmer vyžaduje definíciu pravidiel pre český jazyk (čo znamená nutnosť znalosti gramatiky a češtiny celkovo) a komplexnejšie testovanie pre odhalenie pravidiel, ktoré sa aplikujú nesprávne. Pre vyhľadávanie v produktovej doméne je prijateľnejším scenárom nenájdenie základu slova a ponechanie pôvodného slova (stále je možnosť nájsť čiastočnú zhodu) ako aplikovanie pravidiel, ktoré spôsobia nesprávnu transformáciu a vrátenie nerelevantných dát. Stemmer pre český jazyk (vstavaný, alebo externý) prináša do vyhľadávania ako celku sémantiku daného jazyka. Počas testovania boli implementované varianty jazykových analyzátorov ktoré obsahovali vstavaný stemmer, externý stemmer a varianta bez stemmeru. Z dôvodov a relevantnosti tvarov slov bola varianta bez použitia stemmeru nevýhodná.

Obr. 9: Ukážka definície vstavaného a externého stemmera pre češtinu

Na obrázku 9 možno pozorovať definíciu vstavaného czech_stemmer a externého hunspell_cs_CZ stemmer token filtra, ktorý je definovaný ako typ Hunspell. Hunspell je sada slovníkov, ktorá je verejne dostupná a používa sa v rámci OpenOffice, LibreOffice a pod. Práve tieto slovníky slúžia ako zdroj pre definovanie vlastného, externého stemmera. Inštalácia vlastného stemmera je analogická s inštaláciou synoným, ktorá bola popísaná. Hunspell stemmer pozostáva zo súborov :

- *.dic – obsahuje slová
- *.aff – obsahuje ich možné prefix a sufixy

Súbory sú následne integrované na každý uzol, v rámci zložky hunspell, ktorá je vytvorená v /etc/elasticsearch .

```
- conf
    |-- hunspell
    |    |-- en_US
    |    |    |-- en_US.dic
    |    |    |-- en_US.aff
```

Obr. 10: Umiestnenie Hunspell slovníkov v rámci konfiguračnej zložky uzla

Pomocou atribútu locale, ktorý možno pozorovať na obrázku 9 definujeme, aký typ súboru chceme pre filter použiť (cesta /etc/elastricseatch/hunspell je rozpoznaná automaticky).

Ďalšie token filtre

Okrem stemmera sa v implementácií používajú aj ďalšie token filtre pre úpravu tokenov. Dôležitými sú najmä :

- Unique – zabezpečuje deduplikáciu tokenov na rovnakej pozícií
- Word Delimiter – delí slová na podslová na základe separátorov ako pomlčka, alebo kapitálky v CamelCase slove, slovo „Wi-Fi“ je rozdelené na „Wi“,„Fi“ a slovo „PowerShot“ na „Power“, „Shot“; sú dostupné nastavenia pre zachovanie originálneho slova, spájania číslic a pod.
- Lowercase – prevedie stream na malé písmená, vyhľadávanie je často case - insensitive
- ASCII Folding – rieši diakritiku a špeciálne znaky, ktoré prevádza na ASCII ekvivalent

Súhrn

V implementácií analyzátora bol testovaný vstavaný a externý stemmer spolu s rôznymi kombináciami tokenizéra a ďalších filtrov. Nasledujúca tabuľka zhŕňa najpodstatnejšie kombinácie testovaných analyzátorov.

Názov analyzátora	Tokenizér	Token filter (poradie)	Nasadenie
keyword_hunspell	Keyword	Word Delimiter, Unique, Hunspell (externý), Lowercase, ASCII Folding	Áno
keyword_czstemm	Keyword	Word Delimiter, Unique, CzechStemmer (vstavaný), Lowercase, ASCII Folding	Nie
standard_hunspell	Standard	Unique, Hunspell (externý), Lowercase, ASCII Folding	Áno
standard_czstemm	Standard	Unique, CzechStemmer (vstavaný), Lowercase, ASCII Folding	Nie
white_hunspell	Whitespace	Unique, Hunspell (externý), Lowercase, ASCII Folding	Nie
white_czstemm	Whitespace	Unique, CzechStemmer (vstavaný), Lowercase, ASCII Folding	Nie

Tabuľka 5: Návrh jazykových analyzátorov

Pri testovaní vykazovalo používanie vstavaného stemmera (bez ďalších filtrov) zvláštne výsledky – stemmer orezával konce tokenov. Pre vstupné slovo „mobilný“ bolo výstupným tokenom slovo „mobiln“. Správanie vstavaného českého stemmera neodpovedalo predpokladom pre funkčnosť. V rámci diskusných fór pre Elasticsearch sa nepodarilo dohľadať relevantné informácie o prípadnej oprave a preto implementácia pracuje len s externým Hunspell stemmerom. Whitespace a Standard tokenizéry vykazovali v testovaných prípadoch veľmi podobnú funkcionalitu. Z dôvodu exaktnejšieho definovania a odporúčania dokumentácie sa v implementácii zvolilo používanie Standard tokenizéra. Keyword tokenizér predstavuje alternatívnu variantu k zvolenému Standard tokenizéru a bol zvolený ako ďalší analyzátor pre implementáciu z dôvodu ďalšieho testovania a sledovania funkčnosti pri reálnom používaní IS. Indexy boli vytvorené s 2 verziami analyzátorov pre fulltextové dátové polia, čo predstavuje istú vývojovú variantu. Je možné tak

testovať indexáciu a vyhľadávanie reálnych dát pre oba prípady a prípadne ďalej modifikovať popísane konfigurácie.

2.4.5 Vyhľadávanie

Popísané možnosti a rozhodnutia pre konfiguráciu analyzátorov predstavuje len polovicu implementácie fulltextového hľadania. Druhou komplexnou témou je definícia vyhľadávacích funkcií a procesu. V Elasticsearch sa jedná o množinu dostupných funkcií v rámci Search APIs a Query DSL (Domain Specific Language). Search APIs popisuje všeobecný prístup k hľadaniu v rámci platformy Elasticsearch, Query DSL je jazyk pre tvorbu špecifických dopytov.

Analyzátory sú súčasťou vyhľadávania v čase indexovania a vyhľadávania. Vyhľadávacie API následne umožňujú ďalšie spôsoby hľadania, filtrovania, ovplyvňovania skóre atď.

Vyhľadávanie z pohľadu prístupu delíme na fulltextové dopyty a dopyty na presnú zhodu, označované ako term alebo keyword. Implementácia je zameraná na fulltextové vyhľadávanie a preto nebude pojednávaný druhý spomenutý spôsob, ktorý obsahuje problematiku agregácií a pod. Vyhľadávacie API ako celok definuje 2 základné delenia:
Delenie podľa kontextu :

- kontext vyhľadávania
- kontext filtrovania

Delenie podľa typu klauzuly :

- jednoduché vyhľadávanie
- zložené vyhľadávacie dopyty

Kontext vyhľadávania sa zameriava na relevanciu a kvalitu výsledkov získaných z fulltextového hľadania. V tejto časti je kalkulované skóre, ktoré reprezentuje relatívnu zhodu voči ostatným dokumentom. Kontext filtrovania sa používa na filtrovanie štrukturovaných dát. Výstupom filtrovacieho kontextu je binárna odpoveď – podmienka je splnená alebo nesplnená. Tento kontext nekalkuluje skóre. Príkladom je fulltextový dopyt, ktorý hľadá produkt na základe vstupu „samsung s8“, filtruje však výslednú množinu dokumentov na základe atribútu status, ktorý je v podmienke filtra nastavený na hodnotu „enabled“. Dopyt vyfiltruje dokumenty, kde je podmienka splnená a vypočíta skóre pre množinu vzhľadom na vstup „samsung s8“.

Jednoduché fulltextové dopyty predstavujú fundamentálny prístup k hľadaniu, definujú základné dopyty. Zložené dopyty slúžia ako obal pre skladanie viacerých jednoduchých dopytov a pridávajú možnosť ovplyvňovania a kontrolovania vypočítaného skóre. V implementácií budú popísané niektoré dôležité typy a implementovaných niekoľko typov vyhľadávacieho dopytu. Komplexnosť tejto tematiky je podobná komplexnosti analyzátorov. Existuje mnoho kombinácií a vytvorenie ideálnej varianty spočíva v testovaní a analyzovaní. [9][10]

Relevancia

Relevancia je v rámci Elasticsearch reprezentovaná ako skóre každého dokumentu, jedná sa o kladné desatinné číslo. Relevanciou rozumieme algoritmus, ktorý počíta, ako podobný je obsah fulltextového dátového poľa k zadanému fulltextovému dopytu. Štandardný algoritmus v Elasticsearch pre počítanie podobnosti je TF/IDF (term frequency/inverse document frequency). TF/IDF kalkuluje s faktormi :

- Term frequency (frekvencia výrazu) – koľko krát sa výraz vyskytuje v dátovom poli. Čím viac, tým je záznam relevantnejší.
- Inverse document frequency (obrátená frekvencia dokumentov) – ako často sa daný výraz vyskytuje v indexe. Čím viac, tým je záznam menej relevantný. Výrazy ktoré sa vyskytujú v mnohých dokumentoch majú nižšiu váhu.
- Field-length norm (dĺžka dátového poľa) – čím je dátové pole dlhšie, tým sú výrazy v ňom menej relevantné . Rovnaký výraz je relevantnejší v kratšom dátovom poli.

Relevancia je počítaná každým typom vyhľadávacieho dopytu iným spôsobom. Zložené dopyty umožňujú skladať výsledky relevancie – sčítať, násobiť, vybrať len najlepšiu.

Vyhľadávacie dopyty

Implementácia vyhľadávania začína pri testovaní a analyzovaní jednoduchým typov dopytov, ktorých základný popis je zachytený v tabuľke.

Názov	Popis	Použitie
match	štandardný typ, podporuje fuzzy, proximity (blízkosť)	Áno
match_phrase	podobné match, ale pre presnú zhodu fráz a proximity	Áno
match_phrase_prefix	prefixové hľadanie, podobné match_phrase	Nie
multi_match	hľadanie naprieč viacerými dát. poľami	Áno
common_terms	špecializovaný typ, preferuje jedinečné slová	Nie
query_string	založené na Lucene, podporuje AND, OR, NOT logiku	Nie
simple_query_string	jendoduchšia, robustná varianta query_string	Nie

Tabuľka 6: Typy jednoduchých vyhľadávacích dopytov [10]

match query

Match query je dopyt typu boolean. Znamená to, že vstupný text je analyzovaný a je vytvorený boolean dopyt. Pomocou príznaku operator je možné definovať, či má byť medzi tokenmi (slovami) použitý logický AND alebo OR. Prednastavená hodnota je OR. Match query podporuje fuzzy logiku. Fuzzy logika pojednáva zhodu dokumentov s počítaním určitej úrovne nepresnosti, často sa používa ako faktor ktorý dokáže spracovať preklepy v zadanom vyhľadávacom vstupe.

```
GET /_search
{
    "query": {
        "match" : {
            "message" : "this is a test"
        }
    }
}
```

Obr. 11: Ukážka match dopytu

Match query predstavuje najvšeobecnejší prístup, jeho hlavnou výhodou je robustnosť. Kým iné vyhľadávacie funkcie bežne vykazujú výnimky pri prehľadávaní dátového poľa s nesprávnymi dátami a pod., match query zabezpečuje maximálnu prístupnosť a funkčnosť aj v prípade chybných, či neúplných dopytov a dát.

match_phrase, match_phrase_prefix

Vyhľadávacie dopyty s podobnou funkcionalitou. Hľadajú na základe zhody fráze, čo je určitá podoba hľadania na základe presnej zhody. match_phrase_prefix predstavuje prefixovú variantu, ktorú je možné použiť na implementovanie dopĺňania a navrhovania fráz. Ako bolo popísané v predošlých častiach, Elasticsearch pre takéto účely natívne definuje dátový typ Completion a odpovedajúci typ hľadania.

multi_match

Založené na match query, podporuje dopytovanie viacerých dátových polí. Je možné definovať aké dátové polia sa majú prehľadávať na základe wildcard znaku. Dátovým poliam je možné priradiť váhu, čo sa následne prejavuje na vypočítanom skóre. Definuje 3 typy :

- best_fields – hľadá dokumenty, kde sa našla zhoda v ľubovoľnom zadanom dátovom poli, výsledné skóre je skóre najrelevantejšieho dátového poľa
- most_fields - hľadá dokumenty, kde sa našla zhoda v ľubovoľnom zadanom dátovom poli, kombinuje skóre všetkých dátových polí
- cross_fields – kalkuluje skóre spôsobom, že všetky zadané dátové polia považuje za jeden celok

Zložené dopyty kombinujú spôsoby popísané vyššie. Tabuľka 7 popisuje možnosti zložených dopytov.

Názov	Popis	Použitie
constant_score	vykoná query vo filter kontexte, vracia konštantné skóre	Nie
bool	kombinuje jednoduché query a ich skóre	Áno
dis_max	prijíma viacero query, skóre je z najlepšej vyhľadávacej klauzule	Nie
function_score	umožňuje upravovať skóre na základe funkcií a faktorov	Áno
boosting	rozlišuje kalkulovanie skóre pre kladné a negatívnu zhodu	Nie

Tabuľka 7: Typy zložených dopytov [10]

Vyhľadávacie funkcie je možné kombinovať. Pre vyhľadávanie pomocou viacerých jednoduchých typov je možné použiť bool query a celkové skóre obohatiť o vlastný faktor pomocou function_score.

V tabuľkách 6 a 7 vyššie bolo označené použitie v implementácií. Vybrané typy boli testované a analyzované a odpovedajú potrebám aktuálnej implementácie – vyhľadávania v produktovej doméne. Boli vytvorené dva hlavné typy plnohodnotného fulltextového hľadania a jeden typ pre implementáciu hľadania spôsobom „hľadaj ako píšeš“ (search as you type), v texte často označované ako dopĺňanie a navrhovanie.

```
GET /_search
{
  "query": {
  "function_score": {
    "functions": [
      {
        "field_value_factor": {
          "field": "category_priority_level"
        }
      }
    ],
    "query": {
      "bool": {
        "must": {
          "match": {
            "translate.cs.title.fulltext": {
              "query": "Adidas"
            }
          }}}}}}}
```

Výpis 1: Implementovaný dopyt pre plnohodnotné fulltextové hľadanie, typ 1

```
GET /_search
{
    "suggest": {
    "my-trademark-suggestion": {
        "prefix": "app",
        "completion": {
          "field": "cs.title_analyzed"
      }}}
}
```

Výpis 2: Implementovaný dopyt pre navrhovanie a dopĺňanie

```
GET /_search
{
    "query": {
      "multi_match": {
         "query": "iphone",
         "fields": [
           "name.czech_lowercase^5",
           "name.czech_hunspell^3",
           "phrase.czech_lowercase^3",
           "phrase.czech_hunspell^2"
         ]}}
}
```

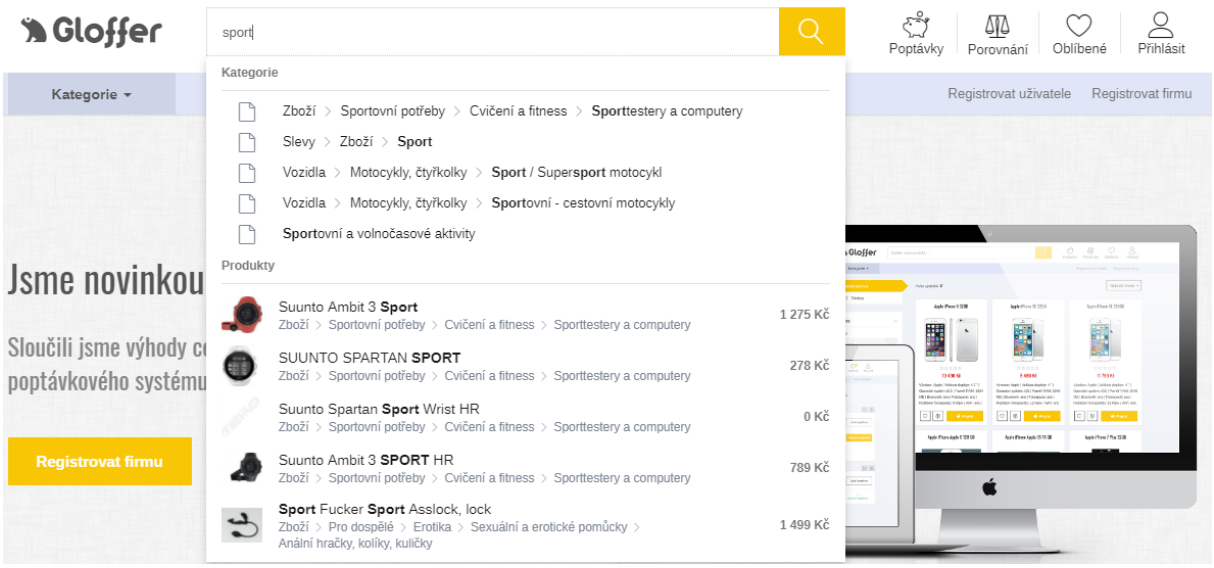
Výpis 3: Implementovaný dopyt pre plnohodnotné fulltextové hľadanie, typ 2

2.5 Nasadenie vyhľadávania

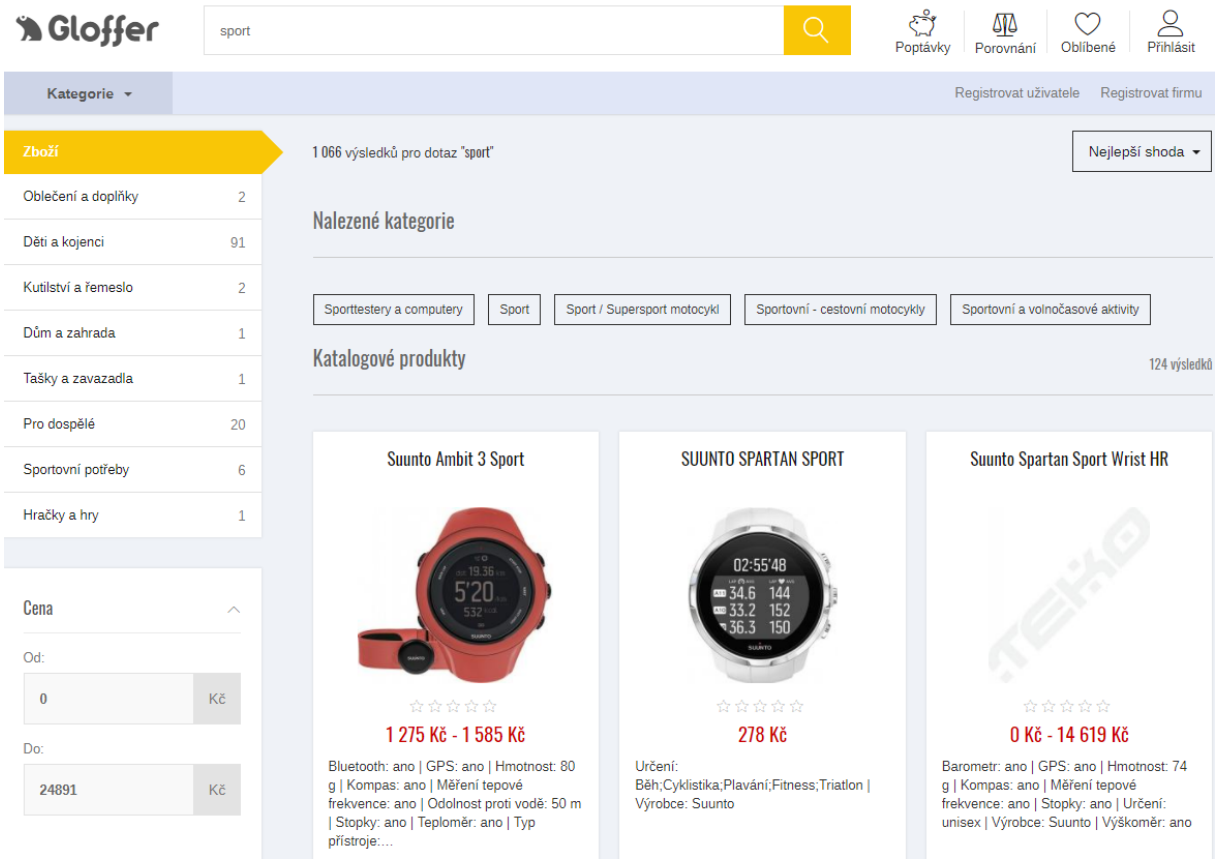
Po vytvorení a konfigurácii clustra, implementovaní jazykových závislostí a vyhľadávacích dopytov je možné celok integrovať v rámci IS, konkrétne v rámci webovej stránky. Implementované vyhľadávanie má ako celok v rámci IS široké uplatnenie, konkrétne :

- doplňanie a navrhovanie na úvodnej stránke
- vyhľadávanie vo zvolenej kategórií, podstránke, vrátane filtrácie (plnohodnotné hľadanie)
- vyhľadávanie v administračnom rozhraní, pre zjednodušenie prístupu

Fulltextové hľadanie je na strane frontendu prezentované pomocou pluginov, ktoré zjednodušujú interakciu s používateľom. Typicky zaobstarávajú intervalové odosielanie vyhľadávacích dopytov (pri navrhovaní fráz), umožňujú zvýraznenie nájdenej zhody a pod. Príkladom takéhoto pluginu je napríklad typeahead. Nasledujúce ukážky prezentujú integrované fulltextové hľadanie vo vyvíjanom IS.



Obr. 12: Navrhovanie výsledkov vyhľadávania na titulnej stránke



Obr. 13: Výsledok plnohodnotného hľadania

3 Predspracovanie obsahu a dátové procesy

Kapitola pojednáva predspracovanie dát a dátové procesy v projekte Gloffer. Problematika nadväzuje na predošlú kapitolu zameranú na fulltextové hľadanie v oblasti tvorby relevantného obsahu fulltextových štruktúr, ich plnenie a aktualizácie. Cieľom je definovanie procesov a implementácia nástrojov, ktoré zabezpečujú migrácie predspracovaných dát.

3.1 Motivácia

Predspracovanie obsahu je technika, ktorej primárnym cieľom je odľahčenie a šetrenie výpočtového výkonu. Vyvíjaný IS pracuje v pozadí s relačnou databázou, konkrétne MySQL 5.4 s InnoDB Engine, ktorá predstavuje hlavné, perzistentné úložisko dát. Relačná schéma obsahuje dátovú doménu v deduplikovanej podobe, rozloženú medzi entity (tabuľky) definovaním vzájomnej kardinality a ordinality. Štruktúrované uloženie dát je ideálne z pohľadu spravovania a manažmentu IS. Z pohľadu vizualizácie dát, alebo inej formy konzumácie obsahu je často potrebné rozložené dáta spojiť do jedného celku, ktorý popisuje skutočnosť a reprezentuje objekty reálneho sveta.

Ideálnym príkladom pre vyvíjaný IS je produkt v rámci produktového agregátora. Produkt je definovaný súborom vlastností a hodnôt, popisnými atribútmi, informáciami o vlastníctve a vzťahu k ďalším entitám. Technicky sa jedná o dátový celok, ktorý navonok vystupuje ucelene, avšak v relačnej schéme je jeho obsah dekomponovaný do viacerých tabuliek. Zostrojenie požadovaného celku – produktu následne odpovedá niekoľkým databázovým operáciám typu spájania tabuliek (JOIN), alebo komplikovanejším výberom dát (SELECT). Frekventované vykonávania náročných operácií naprieč viacerými tabuľkami sa následne prejavuje na výkone celého úložiska, priepustnosti a môže ovplyvňovať ďalšie operácie vykonávané nad relačnou schémou.

Predspracovanie obsahu pristupuje k tomuto problému spôsobom, že požadovaný obsah vygeneruje a uloží s istou časovou platnosťou. Výsledkom je zníženie počtu náročných operácií pre zostrojenie obsahu, keďže je možné použiť platný, uložený súbor dát. Samozrejme dochádza k nárastu uloženej kapacity, avšak pri cenách úložiska na dnešnom trhu ide o preferovanú variantu.

Možnosti predspracovania obsahu v tejto implementácií budú identifikované na základe existujúcich dátových procesov a objektov. Bude pojednávané zostrojenie obsahu, možnosti jeho uloženia a migrácie týchto dát. [6]

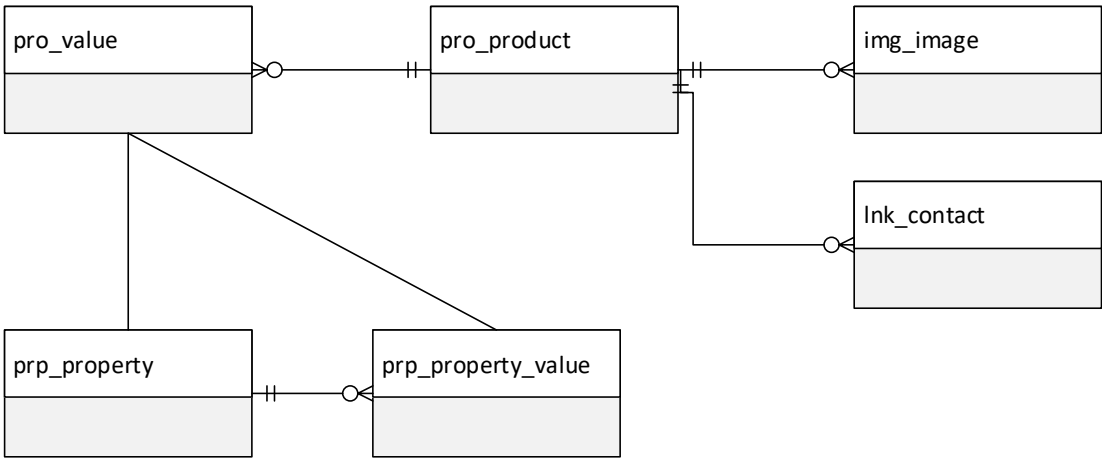
Katalógové a feed produkty

Katalóg produktov je deduplikovaná, unikátna množina produktov, ktoré slúžia ako predpis a agregujú pod sebou reálne predávané produkty firiem. Skladá z informácií ako :

- popisné informácie produktu, preklady popisných informácií,
- vlastnosti produktu

- informácie o cene a dostupnosti,
- kontaktné informácie predajcov,
- alternatívne produkty, príslušenstvo

Nasledujúci obrázok zachytáva katalógový produkt v rámci fragmentu relačnej štruktúry.



Obr. 14: Relačná schéma pre katalógový produkt

Katalógový produkt predstavuje jednu z primárnych entít vyvíjaného IS. Je dôležitý pre proces párovania, kde slúži ako referenčná entita, pre proces vyhľadávania a celkovej vizualizácie v rámci IS.

Proces párovania sa snaží mapovať externé, feedové informácie o produktoch na interné, katalógové produkty v rámci IS. Účelom párovania je agregovanie mnoho feedových informácií pod spoločný základ práve v podobe katalógového produktu. Výsledkom je prehľadný scenár používania IS pre koncového používateľa. Podstatnými faktormi pri agregovaní feedových produktov pod katalógové produkty sú :

- zoznam firiem, ktoré ponúkajú variantu katalógového produktu
- rozsah cien – minimálna a maximálna cena

Proces párovania býva typicky úzko spojený s fulltextovým hľadaním. Feed informácie o produkte bývajú obsiahnuté v niekoľkých dátových poliach. Existujú štruktúrované dátové polia, ako napríklad EAN alebo ISBN, podľa ktorých je možné vykonávať párovanie, avšak vo valnej väčšine sa spracúva názov produktu. Agregátory určujú pravidlá pre názov produktu, definujú poradie hodnôt výrobcu, značky, modelu atď. Proces párovania tak z väčšej, či menšej mieri prebieha na úrovni hľadania zhody textu medzi feedovými názvami produktov a množinou katalógových produktov. Výstupom je množina spárovaných a nespárovaných feedových produktov.

Frekventovanosť procesu párovania odpovedá technickým možnostiam infraštruktúry, minimálna frekventovanosť je však daná spracovaním všetkých feedových produktov za 24 hodín.

Katalógové produkty sú relatívne stabilná množina, z dlhodobého hľadiska má kumulatívny charakter. Nové katalógové produkty pribúdajú zväčša s príchodom nových produktov na trh, prípadne pri spracovávaní novej domény dát.

Proces vyhľadávania má v popisovanom scenári dve funkcie – fulltextové hľadanie pre účely párovania a fulltextové hľadanie pre účely koncového používateľa. Dôležitým poznatkom je, že koncový používateľ z pohľadu katalógových produktov pracuje len s aktívnou množinou. Znamená to, že koncový používateľ vie vyhľadať len tie katalógové produkty, ktoré majú napárovaný aspoň jeden feedový produkt. Proces párovania môže negatívne ovplyvniť mnoho faktorov. Z toho dôvodu je nutné zabezpečiť dostupnosť vyhľadania aj nespárovanej množiny feedových produktov.

Popísaný proces párovania v súvislosti s fulltext hľadaním identifikujú niektoré základné dátové procesy a objekty. Nasledujúca tabuľka stručne zhŕňa hlavné aspekty. Istá analogia týchto dát sa vyskytuje v kapitole Fulltext, vytvorenie indexov odpovedá dátovým procesom. Proces párovania nie je predmetom tejto implementácie, predstavuje len jeden z procesov systému.

Typ dát	Zdroj dát	Rozšírenie dát	Cieľ dát	Účel	Frekventovanosť
Katalógový produkt	Relačná schéma		Elasticsearch index	Párovanie	Ad hoc
Katalógový produkt	Relačná schéma	Agregované ponuky, ceny	Elasticsearch index	Aktívne katalógy IS	Min 1x / 24hod
Feedový produkt	Feed		Elasticsearch index	Nespárované produkty IS	Min 1x / 24hod

Tabuľka 8: Základné dátové procesy

Katalógový produkt vzhľadom na duplicitu výskytu a komplexnosť svojho zloženia predstavuje ideálnu entitu pre predspracovanie. Prístup možno aplikovať aj na ďalšie objekty systému, avšak pre katalógový produkt sa predpokladá najväčšia efektivita v súvislosti s šetrením výpočtového výkonu. Pre feedové produkty možno identifikovať potrebu istého dočasného uloženia v súvislosti s párovaním, agregovaním cien a ponúk a publikovania do fulltextových štruktúr. Implementácia bude zameraná na procesy, predspracovanie a dočasné ukladanie práve pre katalógové a feedové produkty.

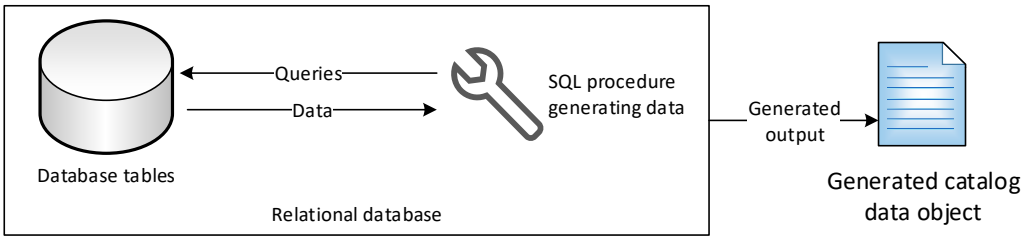
3.2 Generovanie obsahu

Generovanie predspracovaného obsahu je zamerané na katalógový produkt (a iné, štruktúrované entity systému). Feed produkt predstavuje variabilnú entitu, ktorá nie je štruktúrovaná. Feed produkt nie je manažovaný v rámci IS, jedná sa o externú informáciu, ktorá je v informačnom systéme agregovaná. Proces predspracovania pozostáva z fáze generovania a uloženia obsahu. Zdrojom pre generovanie obsahu je relačná databáza MySQL.

Pre generovanie obsahu sú dostupné dve varianty – generovanie pomocou SQL procedúry, alebo generovanie pomocou externého nástroja, programu. Dôležitým faktorom je doba, resp. čas kedy požadujeme mať obsah predspracovaný, alebo vyvolať jeho predspracovanie. Obe varianty majú všeobecne radu výhod a nevýhod. Uložená SQL procedúra však prináša pohodlnejšie

možnosti pre ad hoc generovanie a centrálnu dostupnosť procedúry. Proces generovania môže byť vyvolaný z prostredia administrácie IS, z back-end aplikácie, vyvolaný udalosťou (prekročenie platnosti) a podobne. Použitie externého softvéru prináša ďalšie úlohy, ako zabezpečenie dostupnosti funkcionality (funkcia je dostupná z viacerých miest IS), nasadenie na (virtuálne) zariadenie a iné. Ďalším rozdielom je prenos dát po sieti. Kým SQL procedúra vykonáva svoje dopyty v rámci databázy, externý softvér musí najskôr dáta preniesť po sieti a následne vygenerovať do požadovanej podoby. V prípade existencie databázy a (virtuálneho) zariadenia pre generovania, ktoré sú oddelené v rámci iných častí siete môže ísť o podstatný výkonnostný faktor. Databázové systémy pre uložené SQL procedúry ponúkajú radu optimalizačných možností ako pred-generované SQL dopyty a predgenerované plány vykonávania. Hlavnou doménou pre predspracovanie obsahu sú katalógové produkty. Tieto dáta sú z pohľadu predspracovania takmer nemenné. Po korektnom vygenerovaní dátového celku reprezentujúceho produkt sa predpokladajú len minimálne aktualizácie. [4] [5]

Z dôvodov popísaných v predošlom texte bolo rozhodnuté, že generovanie obsahu pre katalógové produkty bude zabezpečovať uložená SQL procedúra. Tá je následne dostupná iným entitám a častiam systému v prípade potreby, avšak vzhľadom na relatívnu nemennosť predgenerovaných dát je najreálnejší scenár počiatočného, inicializačného vygenerovania a uloženia.



Obr. 15: Schéma generovania katalógového objektu

Katalógové produkty reprezentujú obraz produktov reálneho sveta a ich základná podoba je nemenná. Mierne variabilnou zložkou je väzba na entity vlastníka a plne variabilnou zložkou je minimálna a maximálna cena a ponuky vzhľadom na reálne produkty. SQL procedúra generuje katalógový produkt, ktorého obsahom sú práve statické, nemenné popisné dáta.

Feedové produkty sú predspracované v zmysle párovania. Po procese párovania je pre každý feedový produkt pridelený odpovedajúci katalógový produkt, alebo je produkt nenapárovaný. Napárovaná množina katalógov v rámci feedových produktov predstavuje aktívnu množinu katalógov v IS a tieto dáta sú zdrojom pre následné získanie agregovaných cien a ponúk.

3.3 Možnosti uloženia predspracovaného a dočasného obsahu

Vygenerovaný obsah predstavuje agregované štruktúrované dáta. Po vygenerovaní tento dátový objekt reprezentuje odpovedajúci stav vzhľadom na dáta a závislosti z relačnej schémy, z ktorej

bol vytvorený. Zmena dát, závislostí alebo vzťahov v relačnej schéme (v doméne relevantnej k spracovávaným dátam) znamená, že obsah objektu, ktorý reprezentuje starú verziu neodpovedá novému, reálnemu stavu. Frekvencia zmien na perzistentné, zdrojové dáta udáva dobu platnosti pred-generovaných dát. Sledovanými faktormi pre možnosti uloženia sú:

- formát, typ dát
- frekvencia nutnosti pre-generovania obsahu
- používanie predspracovaného obsahu

Jedným typom predspracovaného obsahu je katalógový produkt, ktorý bol ako celok už detailne popísaný. Tieto dáta reprezentujú produkty reálneho sveta a ich základná podoba je nemenná. Mierne variabilnou zložkou je väzba na entity vlastníka a plne variabilnou zložkou je minimálna a maximálna cena a ponuky vzhľadom na reálne produkty. Predspracovaný obsah katalógového produktu má v rámci IS dve hlavné použitia. Prvým je vytvorenie obsahu full-textových štruktúr pre potreby párovania. Dôležitými časťami sú v tom prípade popisné dáta, ceny a ponuky sú irelevantné. Druhým je vytvorenie aktívnej množiny dostupných katalógových produktov, vrátane agregovaných ponúk a cien. Táto množina musí byť taktiež zaradená do fulltextových štruktúr pre potreby vyhľadávania. Formát takto vygenerovaných sa (všeobecne) označuje pojmom dokument. Implicitne sa jedná o dáta principiálne odlišné od dát v relačnej schéme. Spôsob uloženia a ďalšie operácie nad dátami tohto typu otvárajú tematiku NoSQL databáz.

3.3.1 SQL a NoSQL databázové úložiská

V pozadí každého IS stojí určité riešenie pre uchovávanie dát. Kým v minulosti bolo majoritné používanie výlučne relačnej schémy, s nárastom komplexnosti na prácu s dátami popri klasických relačných databázových systémoch začali vznikať riešenia, ktoré označujeme ako NoSQL. V prvotnej fáze sa SQL a NoSQL riešenia diametrálne odlišovali, pričom za hlavné prvky možno považovať :

- Neexistenciu relácií v NoSQL, s tým súvisiacu referenčnú integritu
- Voľnú schému, resp. neexistenciu štrukturovanej schémy v NoSQL
- Neexistenciu JOIN operácií v NoSQL

Kým relačné, SQL riešenia presne definovali štruktúru ukladania dát v podobe tabuliek, stĺpcov a možnosťou zavádzania vzťahov používaním cudzích kľúčov, NoSQL riešenia umožňovali ukladať dáta do kolekcií, alebo inak nazvanej konštrukcii, ktorá je logickou úrovňou particiovania dát na úrovni SQL tabuľky v predom nedefinovanej schéme, kde každý vložený záznam mohol obsahovať iné dátové polia. Vkladaný záznam sa často označuje ako dokument, keďže ako formát záznamu sa používal JSON, prípadne XML.

NoSQL si našli svoje uplatnenie v rade domén riešenia úloh spojených s ukladaním a dopytovaním dát, kde bolo používanie relačnej schémy (zjednodušene povedané) zbytočne zložité vzhľadom na definíciu relačných schém ako takých. Postupom vývoja, ďalším nárastom komplexnosti a požiadaviek sa začali niektoré SQL prvky prenášať do NoSQL riešení a naopak. Niektoré relačné riešenia integrovali ukladanie a dopytovanie záznamu uloženého vo formáte JSON. Naopak, niektoré NoSQL riešenia zostali svojou podstatou nezmenené. Začali však podporovať používanie syntaxe podobnej SQL, ale taktiež začali vznikať hybridné riešenia, ktoré napríklad integrovali možnosť JOIN operácií. Nasledujúca tabuľka zhŕňa hlavné výhody a nevýhody SQL voči NoSQL.

SQL, výhody a nevýhody	NoSQL, výhody a nevýhody
SQL query jazyk	Flexibilný dátový model
Optimalizované pre veľké množstvo záznamov v tabuľke, indexi	Horizontálna škalovateľnosť
	Ukladanie masívnych data objektov
Transakcie v rámci jedného dotazu	Rýchle query v rámci jednej kolekcie
Široká škála dostupných databáz	HA a partition tolerance
Query dát 1 tabuľky, Query dát viac tabuliek (optimalizácia)	Referenčná integrita dát
Konzistencia dát	Query jazyk(y)
Predefinóvaná a neflexibilná schema	Query medzi viacerými kolekciami
Vertikálna škalovateľnosť	Nezaručená konzistencia

Obr. 16: Porovnanie hlavných prvkov SQL a NoSQL databáz

Kým SQL poskytuje jednoznačný dopytovací jazyk, NoSQL riešenia často vytvárajú vlastné, prípadne čiastočne integrujú nejaké prvky jazyka SQL.

Dopytovanie v rámci jednej tabuľky v SQL býva zvyčajne veľmi rýchle, vďaka používaniu indexov a optimalizáci. Túto vlastnosť však postupne zavádza aj väčšina NoSQL riešení. NoSQL výrazne zaostáva v dopytovaní medzi viacerými kolekciami, kde buď nie je dostatočná možnosť optimalizácie, alebo riešenie takúto vlastnosť ani nepodporuje. SQL riešenie ponúkajú možnosti optimalizácie, vďaka ktorým je možné dosiahnuť výkonné dopytovanie naprieč viacerými tabuľkami. Hlavná výhoda NoSQL riešení je v ich škálovateľnosti.

Databázu je možne škálovať naprieč niekoľko serverov, vytváraním clustra s uzlami, čo nazývame horizontálne škálovanie. SQL databáza de-facto beží na jednom servery. Škálovanie relačnej databázy teda znamená zvyšovanie výkonu daného servera, čo označuje ako vertikálne škálovanie.

Technický výpadok v rámci NoSQL zvyčajne neovplyvňuje funkčnosť, práve vďaka horizontálnej škálovateľnosti. Pád servera s relačnou schémou, resp. jeho nedostupnosť býva väčšinou zásadný problém, pre ktorý existujú možnosti riešenia ako mirroring a používania niekoľkých

pripojení, nejedná sa však o tak flexibilné riešenie ako ponúka NoSQL. Referenčná integrita dát ja plne naklonená relačným databázam.

V súčasnosti existujú desiatky konkrétnych NoSQL riešení. Ich základné rozdelenie je podľa typu :

- key-value (kľúč-hodnota) orientované databázy
- dokumentové databázy
- column-wide (stĺpcové) databázy
- grafové databázy

Rôzne typy NoSQL databáz odpovedajú rôznemu formátu a uloženiu záznamov, zjednodušene možno toto rozdelenie chápať ako rozdielne úrovne granularity ukladanej dátovej jednotky. Pre túto implementáciu práce s predspracovaným obsahom sú najpodstatnejšie dokumentové databázy.

Dokumentové databázy sa vyznačujú adaptovaním key-value techniky s pridaním komplexnosti v zmysle dokumentu ako hodnoty uloženej pod kľúčom. Z popísaných dátových štruktúr a objektov pre ktoré je pojednávané predspracovanie je možné pozorovať, že vygenerovaný obsah predstavuje dokument, kľúč môže pri generovaní odpovedať napríklad primárnemu kľúču. [12] [13]

V nasledujúcom texte budú popísané niektoré technologické možnosti ukladania predspracovaného obsahu, pre NoSQL špeciálne typy dokumentových databáz.

MySQL JSON

Relačná databáza MySQL od verzie 5.4 natívne podporuje ukladanie a dopytovanie záznamu vo formáte JSON, konkrétne zaviedli dátový typ JSON. Možno tak pozorovať zavádzanie prvkov typických pre NoSQL do relačnej SQL databázy. MySQL JSON podporuje radu funkcií pre tvorbu JSON objektu, aktualizáciu, objektu. Dostupné sú aj základne funkcie pre porovnávanie, určovanie podmienok a agregovanie hodnôt z množiny JSON objektov. MySQL JSON prináša možnosť základnej práce s JSON objektami, avšak NoSQL poskytujú (natívne) omnoho komplexnejšie možnosti. [14]

MongoDB

MongoDB je jedna z najpoužívanejších open-source dokumentových NoSQL databáz. Záznamy ukladá vo formáte BSON, ktorý je veľmi podobný formátu JSON. BSON predstavuje binárne kódovanú serializáciu JSON dokumentu, pridáva podporu dátových typov ako napríklad dátum.

MongoDB je distribuovaná databáza, cieleným aspektom je však konzistencia dát. Poskytuje možnosť horizontálneho škálovania prostredníctvom Shardovania. Shardovanie predstavuje distribúciu dát a záťaže medzi viacero serverov. Jedná sa o mechanizmus, ktorý je potrebný

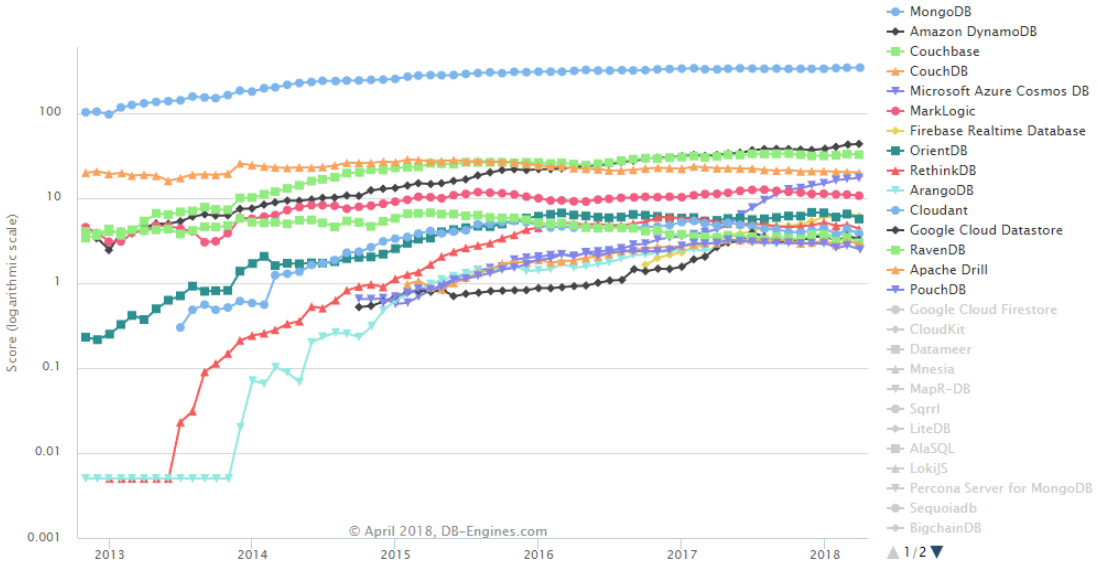
v prípade, že spracúvame veľké datasety a narážame na obmedzenie pri použití a škálovaní jedného servera. Infraštruktúra, ktorá poskytuje horizontálne škálovanie pomocou shardovanie nazývame shardovaný cluster. Replikácia v rámci MongoDB znamená ukladanie kópií dát na rôzne servery. Redundancia dát, ktorá v tomto prípade nastáva má výhody : zaisťuje dostupnosť dát v prípade výpadku niektorých zo serverov, zaisťuje bezpečnosť pred stratou dát v prípade kritických chýb, nie je potreba žiadnej údržby v zmysle znovu-zostavenia dát a navyšuje výkon čítania dát v prípade potreby. Poskytuje taktiež kvalitné možnosti agregácií, ad-hoc dopytov či vytvárania indexov.[15]

CouchDB

Apache CouchDB je široko používaná, open-source dokumentová databáza. Je vyvinutá v jazyku Erlang a ukladá záznamy vo formáte JSON. Poskytuje ACID sémantiku. Databázu je možné dopytovať priamo pomocou Javascriptu, CouchDB je orientovaná na protokol HTTP a RESTful prístup vo svojom jadre. Taktiež podporuje možnosť škálovania naprieč servermi a vytváranie replík. [16]

Couchbase

Couchbase (označované aj Couchbase Server) je open-source dokumentová a key-value databáza. Umožňuje ukladať ľubovoľné typy binárnych dát ako aj JSON dokumentov. V súvislosti podobného názvu s CouchDB nutno podotknúť, že historicky sa niektoré časti vývoja týchto databáz prelínali, avšak v súčasnosti sa jedná o samostatné, odlišné riešenie. Couchbase poskytuje rôzne možnosti dopytovania : pomocou N1QL (jazyk veľmi podobný SQL), pomocou pohľadov (view, vrátane multidimenzionálnych a geografických dát) a pomocou key-value hľadaní. Couchbase databáza je v základe navrhnutá ako cluster skladajúci sa z uzlov, má vstavaný systém cachovania. [17]



Obr. 17: Trendy dokumentových NoSQL databáz podľa webu db-engines.com

Súhrn

Generáciu obsahu predspracovaného katalógového produktu zabezpečuje SQL procedúra. Obsah tvoria z valnej väčšiny statické popisné dáta. Vygenerovaná množina katalógových produktov SQL procedúrou slúži ako základ pre ďalšie dátové procesy ako plnenie fulltextových štruktúr. Keďže relačná databáza MySQL podporuje uloženie dátového typu JSON, kombinácia SQL procedúry a uloženia v MySQL JSON dátovom type sa javí ako ideálna varianta.

Vygenerovaním a uložením v rámci relačnej schémy tak vznikne množina relatívne nemenných dát. Katalógový objekt je komplexná entita a vygenerovanie požadovanej podoby môže byť pre objemnú množinu časovo a výpočtovo náročná operácia. Inicializačné vygenerovanie a uloženie by tak predstavovalo vytvorenie perzistentnej množiny, ktorá sa bude následne používať pre čítanie ďalšie spracovanie, mimo relačnej schémy. Taktiež je zjednodušená záloha takýchto dát pri vytváraní zálohy relačnej databázy. Z popísaných dôvodov je vygenerovanie a uloženie katalógových produktov a dát analogickej charakteristiky manažované v rámci relačnej databázy MySQL, resp. MySQL JSON.

Feedové produkty do IS prichádzajú v zmysle dokumentov a predstavujú externú informáciu, ktorú IS agreguje vo svojej logike. Množina feed produktov je použitá pre proces párovania a následného agregovania cien a ponúk. Logika párovania nie je predmetom tejto implementácie a v rámci vývoja IS je vyvíjaná ďalšími členmi tímu. Množina feedových produktov kompletne vzniká a zaniká v intervalom určeným v rámci procesov IS. Možno pozorovať určité pravidelné obmeny stiahnutej množiny dát, potrebu aktualizácie niektorých hodnôt naprieč produktami (záznamami, dokumentami) pre integráciu procesu párovania a následnej agregácie požadovaných dát. V tomto prípade sú celý proces a dáta oproti katalógu omnoho variabilnejšie a požaduje

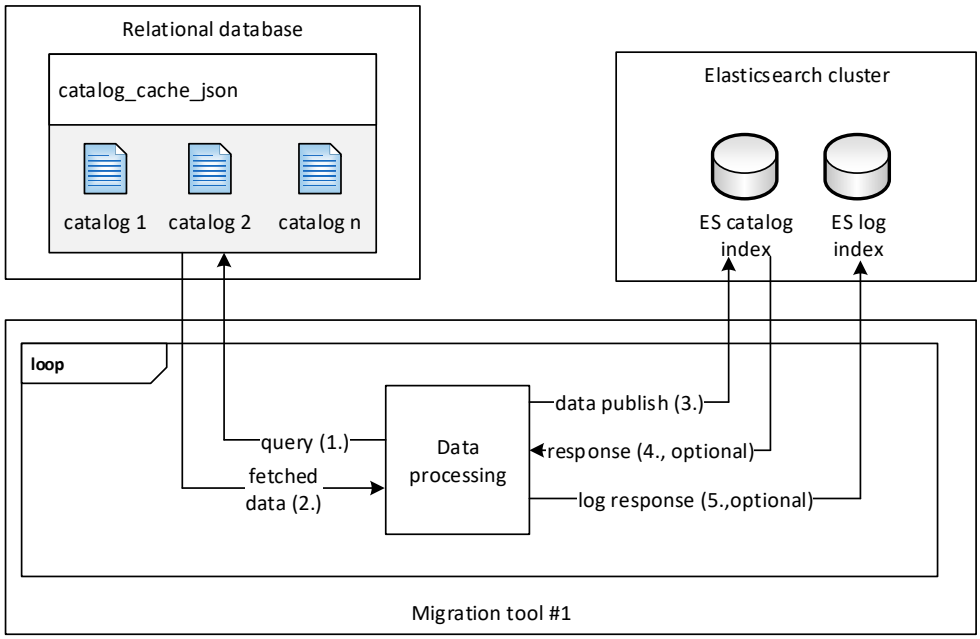
sa širšia funkcionalita nad dokumentovým záznamom. Tieto dáta nie sú vhodné pre zaradenia do štruktúrovanej, relačnej schémy, keďže takéto uloženie neprináša žiadne benefity, dátový typ JSON je nedostatočný z pohľadu agregácií. Pre feedové produkty je vhodnejším variantom práve NoSQL riešenie, konkrétne v podobe dokumentovej NoSQL databázy. Všetky popísane NoSQL riešenia poskytujú kvalitné možnosti pre požadovanú implementáciu. Pre implementáciu nad funkcionalitou feed produktov bolo zvolené riešenie MongoDB. Vzhľadom na objem feedových produktov, ktoré sa pohybuje rádovo v sto tisícoch, požadovanú funkcionalitu rovnako splní aj Couchbase alebo CouchDB. Výber bol ovplyvnený najmä určitými vedomosťami členov týmu o práci s MongoDB. Populárnosť v zmysle jednej z najpoužívanejších NoSQL dkumentových databáz prináša výber medzi IDE nástrojmi, širokú a kvalitnú dokumentáciu a najmä množstvo reálne riešených úloh naprieč diskusnými fórami a podobne. Získané vedomosti pri implementácií predstavujú všeobecné vedomosti pri práci s dokumentovými databázami.

3.4 Návrh a implementácia nástroja pre migráciu predspracovaných dát do fulltextových štruktúr

Hlavnú funkcionalitu v súvislosti s predspracovanými dátami je plnenie fulltextových štruktúr. Sú definované dva hlavné scenáre :

- inicializačná migrácia katalógu produktov do fulltextových štruktúr pre účely párovania
- pravidelná migrácia spárovaných katalógových produktov do fulltextových štruktúr (aktívna množina katalógov) a pravidelná migrácia nespárovaných feedových produktov do fulltextových štruktúr (aktívna množina nespárovaných produktov)

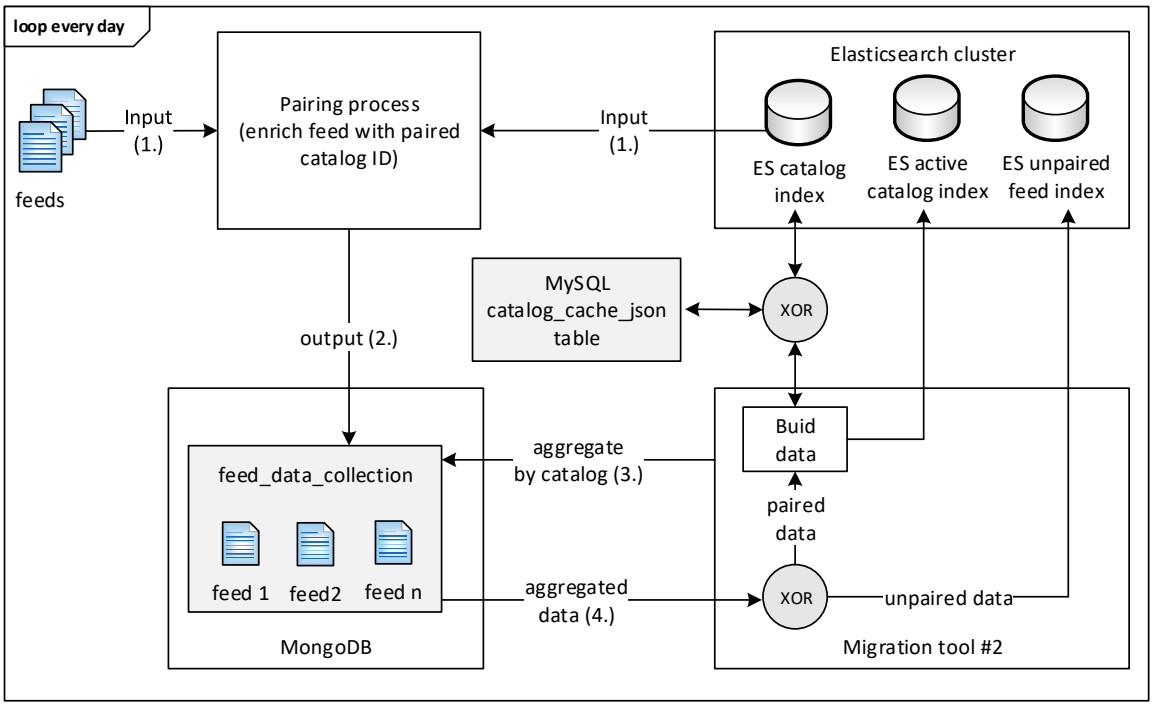
Proces migrácie spárovaných a nespárovaných objektov do fulltextových štruktúr je popísaný v spoločnom bode, pretože dané operácie navzájom súvisia. Návrh procesu migrácie katalógov možno popísať nasledujúcim diagramom.



Obr. 18: Diagram procesu migrácie množiny katalógov

Diagram zachytáva hlavné oblasti systému, ktoré sa podieľajú na procese migrácie. Predpokladom sú hotové, vygenerované JSON katalógové objekty uložené v odpovedajúcej tabuľke (catalog_cache_json). Záznamy sú v cykle čítané, spracované a publikované do fulltextových Elasticsearch štruktúr. Operácia spracovania odpovede je nepovinná. V diagrame sa log záznamy propagujú do odpovedajúceho Elasticsearch indexu. Dôvodom pre toto rozhodnutie je nezaťažovanie relačnej schémy pre operácie vkladania a aktualizovania, podstatný je maximálny výkon v oblasti čítania. Cieľom tohoto procesu je migrácia veľkého počtu katalógov, konkrétne hodnoty boli popísané v časti Fulltext, rádovo sa jedná o desiatky miliónov záznamov. Popísaný proces má zmysel pre inicializačnú migráciu a následné viac alebo menej adhoc aktualizácie týchto dát.

Druhým typom migrácie sú pravidelné (denné) procesy definované párovaním feed a katalógových produktov. Nasledujúci diagram zobrazuje proces publikácie spárovaných a nespárovaných produktov do fulltextových štruktúr. Logika spracovávania log záznamov v diagrame nie je vyobrazená, analogicky k predošlému diagramu možno však log záznamy spracovať do odpovedajúceho Elasticsearch indexu.



Obr. 19: Diagram budovania aktívnej množiny produktov

Vyhotovenie množiny spárovaných katalógov kombinuje dáta z agregačnej funkcie v rámci feed produktov a zdrojových predspracovaných katalógových dát. Tie možno získať z relačnej schémy alebo z Elasticsearch indexu, kde je propagovaná ich kópia.

Implementácia podľa vyššie navrhnutej logiky predstavuje vývoj dvoch nástrojov. Nástroje je možné implementovať v každom populárnejšom jazyku, keďže pre použité úložiská MySQL, MongoDB a Elasticsearch sú dostupné knižnice a konektory pre prácu s dátami. Pre vývoj nástrojov bol zvolený jazyk Java. Dôvodom je začlenenie do back-end infraštruktúry projektu. Je tak možné využiť niektoré konštrukcie naprieč vyvíjanými nástrojmi a aplikáciami.

Migrácia množiny katalógov

Jedným z dôležitých faktorov je možnosť škálovania výkonu migrácií. V prostredí Java je jednou z možností implementácia viac-vláknových aplikácií. V prípade migrácie katalógových objektov je tento faktor dôležitý v súvislosti s testovacími verziami. Zaradenie do fulltextových štruktúr je permanentné voči nastaveným analyzátorom a procesom spracovania textu pre fulltextové hľadanie. V prípade nevhodnej konfigurácie je nutné vykonať re-import dát. Zdrojom je relačná tabuľka, ktorá obsahuje predspracované informácie katalógových objektov. V implementácií boli testované dva scenáre viac vláknového spracovania týchto dát :

- proporcionálne rozdelenie tabuľky, každé vlákno spracúva určitú časť

- sekvenčné čítanie hlavným vláknom a rozdelenie práce medzi ďalšie vlákna, asynchrónne operácie

Cieľom týchto dát sú fulltextové štruktúry – dátové uzly Elasticsearch clustra. V testovacom nasadení sa pre obe implementácie spracúvala aj odpoveď, log o operácií indexovania. Tento faktor môže výrazne ovplyvniť výkon, avšak z pohľadu vývoja je dôležité odhalenie chybných dát a iných problematických častí. Popísaný prvý a druhý spôsob migrácie dát odpovedajú chronologickému vývoju.

Prvý spôsob bol niekoľkokrát testovaný, avšak pravidelne dochádzalo k zahlcovaniu Elasticsearch clustra. Čítanie z relačnej schémy dosahovalo vysokú priepustnosť. Každé vlákno vo veľmi krátkych intervaloch obdržalo množinu dokumentov, ktoré sa snažilo spracovať pomocou konštrukcií dostupných v rámci softvéru pre Elasticsearch. Cieľom bolo nájsť pomer medzi počtom vlákien a veľkosti selektovanej množiny v rámci vlákna. Zahltenie služby Elasticsearch nastávalo z pohľadu počtu požiadaviek a bolo zdržované o spracovanie log záznamov. V prípade zahltenia bolo následne nutné znovu spustiť aplikáciu – reinicializovať objekty, ktoré dopytovali službu a pre zabezpečenie korektného pokračovania spustiť vlákna na pozíciách, kde skončili v predchádzajúcom spustení. Pri hľadaní vhodného pomeru spracovaných dát je jednou z variabilných zložiek aj počet vlákien. Táto zmena v súvislosti s reinicializáciou aplikácie po zahltení a nastavenie nových pozícií vlákien smeroval k tomu, že celý proces sa javil veľmi nejednoznačne. Po testovaní sa pristúpilo k inému spôsobu, ktorý dokáže danú množinu migrovať s vyššou mierou stability s možnosťou dynamického škálovania počtu vlákien a jednoduchším manažmentom.

Druhá verzia nástroja pristupuje k relačnej tabuľke s JSON záznamami sekvenčne. V rámci hlavného vlákna je selektovaná množina dokumentov, ktorá je následne spracovávaná ďalšími vláknami zabezpečujúcimi publikáciu do Elasticsearch štruktúr. Hlavné vlákno tak neustále vytvára prácu pre publikujúce vlákna. Tento proces je jednoznačný z pohľadu ktoré dáta boli z MySQL databázy selektované, je možné dynamicky meniť pomer počtu selektovaných dát a bežiacich vlákien. Implementácia druhým spôsobom predstavuje stabilnejšiu variantu v zmysle vytvárania relevantného počtu index požiadaviek na dátové uzly, ide o preferovanú variantu.

Konkrétne použité konštrukcie pre prácu s dátami v rámci platformy Java a implementovaných nástrojov sú :

- MySQL konektor, oficiálny JDBC driver
- Elasticsearch Java Bulk API, Bulk Processor
- ThreadPoolExecutor, správa vlákien

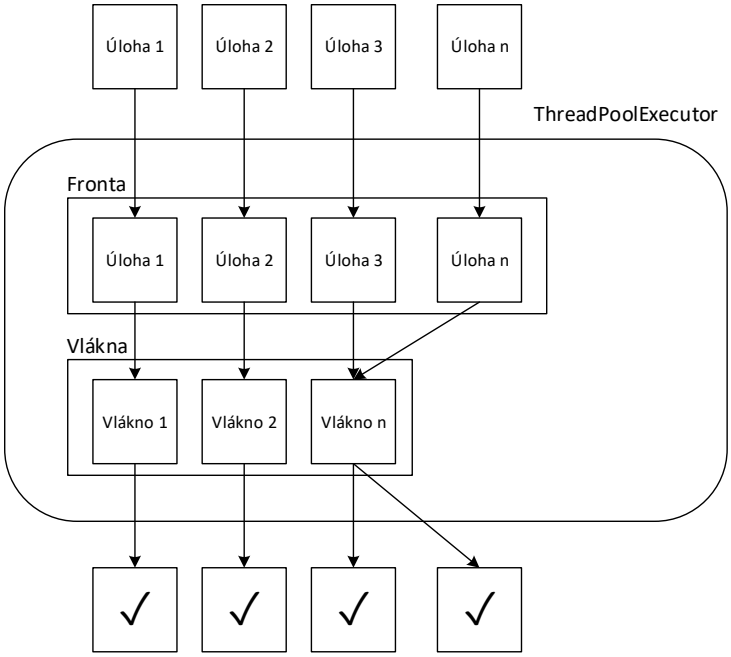
Hlavné vlákno inicializuje tieto konštrukcie a získava dáta pre následné vláknové spracovanie. Samotnú publikáciu zabezpečuje objekt Bulk Processora. Bulk Processor zodpovedá za prijímanie index a update požiadaviek, ktoré po prekročení definovanej kvóty zasiela na odpovedajúce uzly. Kvóta môže byť definovaná počtom požiadaviek, dátovou veľkosťou požiadaviek,

časovým intervalom, alebo kombináciou všetkého. Bežiace vlákna obsahujú referenciu na objekt BulkProcessora. Procesor poskytuje možnosť implementovania callback funkcií beforeBulk a afterBulk, v rámci ktorých je možné vykonať dodatočné úpravy pred odoslaním požiadaviek a analogicky získať odpoveď po ich odoslaní a vykonaní. Callback funkcie umožňujú spravovať logovanie chybných dokumentov, identifikujú situácie zahltenia a iných možných chýb.

```
BulkProcessor bulkProcessor = BulkProcessor.builder(restHighLevelClient::bulkAsync, new
    BulkProcessor.Listener() {
        @Override
        public void beforeBulk(long executionId, BulkRequest request) {
            LOG.info("[BULK EXECUTION] Executing bulk [{}] with {} requests",
                executionId, request.numberOfActions());
        }

        @Override
        public void afterBulk(long executionId, BulkRequest request, BulkResponse response) {
            LOG.info("[BULK SUCCESS] Bulk [{}] completed in {} milliseconds",
                executionId, response.getTook().getMillis());
        }

        @Override
        public void afterBulk(long executionId, BulkRequest request, Throwable failure) {
            LOG.error("[BULK EXCEPTION] Failed to execute bulk", failure);
        }})
            .setBulkActions(processorBulkDocs)
            .setBulkSize(new ByteSizeValue(processorBulkSizeMB, ByteSizeUnit.MB))
            .setFlushInterval(TimeValue.timeValueSeconds(60))
            .setConcurrentRequests(1)
            .setBackoffPolicy(BackoffPolicy.exponentialBackoff(TimeValue.timeValueMillis(50), 3)
                )
            .build();
```

Výpis 4: Inicializácia BulkProcessora a callback funkcií

V implementácií bol pre vytváranie a správu vláknového spracovania použitý wrapper ThreadPoolExecutor. Táto konštrukcia prijíma úlohy v podobe objektov implementujúcich rozhranie Runnable. Jedná sa o implementáciu a integráciu návrhového vzoru producent-spotrebiteľ.



Obr. 20: Schéma vlákien a fronty ThreadPoolExecutora [18]

Tvorba aktívnej množiny produktov vo fulltextových štruktúrach

Implementácia druhého nástroja operuje nad podstatne menšou množinou dát. Viac vláknové spracovanie možno použiť analogicky ako v predošlom nástroji, avšak výkon tejto aplikácie nebude tak efektívny vzhľadom na réžiu pre správu vlákien. Nástroj získa agregované informácie z feedových produktov uložených v MongoDB. Agregačná funkcia agreguje dáta podľa ID katalógu v rámci feed produktov (id katalógu predstavuje informáciu o napárovanom katalógu, ide o výstup z operácie párovania), agregované hodnoty sú minimálna cena, maximálna cena a zoznam feed produktov (ponúk) pre daný katalóg.

```
Document match = new Document("$match", new Document("mapping.pro_catalog_id",
    catalogMatchPart));
Document group = new Document("$group", new Document("_id", "$mapping.
    pro_catalog_id")
        .append("min", new Document("$min", "$frontend.pricing.price_vat"))
        .append("max", new Document("$max", "$frontend.pricing.price_vat"))
        .append("offers",
                new Document("$push",
                        new Document("fir_firm_id", "$unique.fir_firm_id")
                                .append("product_url", "$unique.product_url")
                                .append("shp_shop_id", "$unique.shp_shop_id")
                                .append("shp_feed_id", "$unique.shp_feed_id")
                                .append("currency", "$frontend.currency")
                                .append("language", "$frontend.language")
                                .append("pricing", "$frontend.pricing")
                                .append("delivery", "$frontend.delivery")
                )
        )
);

// construct pipeline
List<Document> pipeline = Arrays.asList(match, group);

// do aggregation query
AggregateIterable<Document> iterable = mongoDatabase.getCollection("feed_collection").
    aggregate(pipeline);
```

Výpis 5: Funkcia agregujúca hodnoty vo feedových produktoch v MongoDB

Ukážka kódu vytvára agregačnú pipeline, ktorá je vstupom do agregačnej funkcie zaslanej na MongoDB kolekciu. V ukážke možno pozorovať použitie vstavaných agregačných funkcií min a max a vytvorenie množiny ponúk, ktorá sa skladá z ID firmy, ID eshopu, ceny, meny, jazyka a spôsobe doručenia produktu. Nasledujúci obrázok demonštruje výstup tejto funkcie.

Obr. 21: Výstup agregačnej funkcie

Získané ID katalógu slúži ako kľúč k získaniu predspracovaného obsahu. Zdrojom môže byť Elasticsearch Index ktorý slúži pre procesy párovania, alebo uložený JSON záznam v databáze MySQL. Nástroj následne kombinuje získané dáta s agregovanými hodnotami, vytvára obsah, ktorý je následne indexovaný do odpovedajúceho indexu. Feedové produkty, ktoré nebolo možné napárovať neobsahujú pridelené ID katalógu, buď zostáva toto dátové pole označené ako null, alebo obsahuje špeciálny príznak, napr. -1. Agregačná funkcia môže buď eliminovať tieto hodnoty a dohľadať ich dodatočne podľa príznaku, alebo produkty zagregovať podľa príznaku. Scenáre sú ekvivalentné.

Popísaný scenár s každou iteráciou vytvára nový index a dáta sú indexované nanovo. Tento prístup predstavuje veľmi jednoduchú variantu vytvorenia požadovanej množiny produktov v rámci fulltextových štruktúr. Opačným prístupom je kontrola existencie záznamu a aktualizácie cien a ponúk. Táto varianta prináša viac réžie na programátorskú časť a z pohľadu výkonu je nutné oproti vloženiu záznamu kontrolovať jeho jednotlivé časti. Prístup vytvárania nového indexu je veľmi jednoducho implementovateľný pomocou konštrukcie Alias v rámci platformy Elasticsearch. Alias umožňuje prideliť indexu názov, ktorým je možné index reprezentovať (s aliasom je možné index dopytovať, indexovať, mazať a pod.). V názve nového indexu tohoto scenára je obsiahnutý dátum a čas (pre potreby odlíšenia indexov). Bez možnosti aliasu by museli konzumenti obsahu vedieť alebo získať presný názov indexu. V prípade ak by bol nový index vytvorený nesprávne z dôvodu chyby, výpadku atď., logika používania funkčného, záložného indexu by bola ešte zložitejšia. Pridelením aliasu indexu vzniká pre konzumentov jednoznačná informácia o názve, ktorý majú používať. Alias môže byť v prípade chyby pridelený staršiemu indexu, z pohľadu konzumenta sa však nič nemení. Z pohľadu implementovaného nástroja tak po vytvorení nového indexu a kontrole jeho stavu nastáva atomická operácia odobrania aliasu starému indexu a pridelenie aliasu novému indexu. Jednou operáciou je tak pod definovaným aliasom reprezentovaný nový index.

```
POST /_aliases
{
"actions" : [
  { "remove" : { "index" : "catalog_active_5_3_2018_12:25", "alias" : "catalog_active" } },
  { "add" : { "index" : "ctalog_active_6_3_2018_12:25", "alias" : "catalog_active" } }
  ]
}
```

Výpis 6: Atomický príkaz odobratie a nastavenie aliasu

Ukážka demonštruje odobranie aliasu a pridelenie novému indexu. Operácia je v rámci Elasticsearch definovaná ako atomická, nie je tak nutné riešiť veľmi krátky interval kedy alias nie je pridelený žiadnemu indexu.

3.5 Testovanie a nasadenie

Vyvinuté Java aplikácie boli v podobe jar súborov nasadené na virtuálne zariadenia v rámci infraštruktúry IS. Virtuálne zariadenie sú vytvorené inštancie počítačov, serverov v rámci serverovej virtuálnej farmy, ktorým sú pridelené relevantné zdroje (pamäť, RAM, procesorový výkon), operačný systém je Debian. Nástroj, ktorý bol implementovaný pre budovanie aktívnej množiny produktov bol začlenený do celkového procesu, ktorý obsahuje proces párovania a ďalších podporných funkcií. Obe aplikácie sú implementované spôsobom, že pre svoj štart požadujú ako argument cestu ku konfiguračnému súboru. Konfiguračný súbor definuje s akou databázou program pracuje, aké sú limity selektovania dát, určuje počet vlákien a pod. Migrácia produktového katalógu predstavuje program, ktorý je spúšťaný ad-hoc spôsobom. Najjednoduchším spôsobom pravidelného budovania aktívnej množiny (druhý nástroj) je zaradenie implementovanej aplikácie do zoznamu CRON úloh.

```
# m h  dom mon dow   command
00 11 * * * cd /data/importer && java -cp importer.jar cz.gloffer.backend.engine.mongo2es.Mongo2ES config.yml
```

Obr. 22: Defínicia spustenia aplikácie ako CRON úlohy

Sledovanie procesov implementovaných aplikácií je možné pomocou vstavaných a vlastných log správ, efektívnejším nástrojom je však Kibana. Kibana je plugin v rámci platformy Elasticsearch. Umožňuje vykonávať monitoring indexovania dokumentov, latencie indexovania, vyhľadávania dokumentov, latencie vyhľadávania a iné. Vizuálne sa jedná o webové rozhranie, ktoré ďalej poskytuje developerský prístup a využívanie REST API. Kibana taktiež umožňuje široké analytické možnosti. Pre migráciu produktového katalógu sú z pohľadu testovania výkonu podstatné konfigurácie :

- počet vlákien

- počet selektovaných dát
- nastavenie Bulk Processora

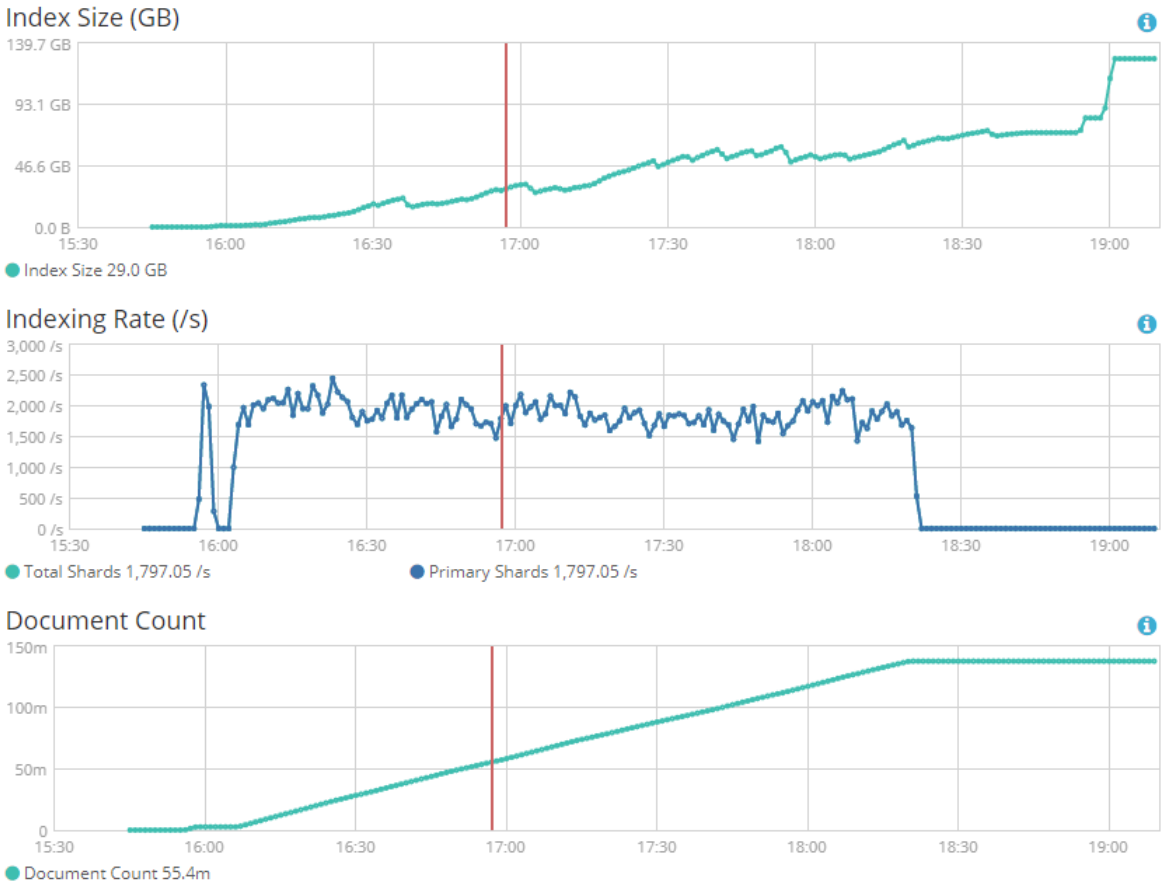
Z dokumentácie a diskusných fór je odporúčané nastavenie kvóty Bulk Processora (kvóta v zmysle naplnenia podmienok, kedy majú byť nakumulované požiadavky odoslané na spracovanie) na 15MB. Migrácia bola testovaná s 2,4 a 8 vláknami, selektované dáta (ktoré boli distribuované do vlákien) sa pohybovali od 2500 do 10 000. Na proces migrácie majú samozrejme vplyv aj ďalšie faktory, ako vyťaženie a priepustnosť zdrojovej relačnej tabuľky a vyťaženosť Elasticsearch clustra. Migrácia bola testovaná v podmienkach, kedy dané služby boli vyťažované minimálne. Jedná o ad-hoc operáciu, takže dopad týchto faktorov možno považovať za irelevantný. Rýchlosť plnenia Bulk Procesora požiadavkami je daná práve počtom vlákien a selektovanou množinou. Pri testovaní bolo sledované, že časť vytvárania požiadaviek je možné variabilne meniť, avšak celkový výkon je pri migrácie množiny o veľkosti niekoľko desiatok miliónov záznamov daný hlavne priepustnosťou Elasticsearch vrstvy. Priepustnosť v tomto prípade určuje :

- počet dátových uzlov
- nastavenie indexu – počet shardov a replík

Väčší počet dátových uzlov znamená širšie rozloženie záťaže. Počet shardov a replík, ktoré sú definované pre index sú jedným z hlavných faktorov ovplyvňujúcich výkon. Kým počet shardov nie je možné meniť a vyplýva z návrhu indexu v clustry, počet replík je možne dynamicky nastavovať. Pre proces hromadnej migrácie sa osvedčila technika vypnutia, resp. nepoužívania replík. Tento prístup znamená, že v pozadí nie sú dáta kopírované do replík, čo indikuje nižšie vyťaženie disku a tým pádom vyššiu priepustnosť pre sledované dáta. Po migrácií je možné nastaviť požadovaný počet replík a Elasticsearch v pozadí začne vykonávať alokáciu a kopírovanie dát. Tento proces môže trvať pomerne dlhý čas, avšak z pohľadu celej migrácie sú už dáta zaradené v indexe (je možné ich používať) a čítanie z relačnej tabuľky , ako aj všetky prenosy po sieti trvali kratší čas.

Samozrejmosťou je určitý limit vzhľadom na hardvér. Pri testovaní migrácie, ktorá bola stabilná počas celej doby, sa priepustnosť pohybovala od 1800 indexovaných záznamov za sekundu po približne 4000 vložených záznamov za sekundu. Pri testovaní s nastaveným počtom replík bola priemerná hodnota približne 2500 indexovaných záznamov za sekundu. Scenár vypnutia replík a nastavenia počtu až po migrácií dosahoval približne priemerne 3500 indexovaných záznamov za sekundu. Stabilným nastavením z pohľadu počtu vlákien a selektovaných dát sú 4 vlákna a sekvenčné selektovanie 4000 záznamov.

Pre pravidelné budovanie aktívnej množiny sú výkonnostné testovania irelevantné. Počty indexovaných záznamov sa pohybujú rádovo v desiatkach až sto tisíc záznamov. Spracovanie takejto množiny dvoma alebo štyrmi vláknami trvá rádovo minúty.



Obr. 23: Graf z procesu migrácie katalógovej možiny z prostredia Kibana

4 Webové služby, aplikačné programové rozhranie

Kapitola pojednáva možnosti implementácie API v infraštruktúre vyvíjaného IS. Implementácia funkčného API môže predstavovať komplexnú prácu, a preto boli v tomto prípade kompetencie rozdelené medzi 3 časti - implementáciu API z technologického pohľadu, vrátane zabezpečenia (predmet tejto kapitoly), implementáciu dátovej vrstvy a konzumovanie (testovanie) API z externého prostredia.

4.1 Analýza súčasných riešení

Rozmach softvéru a informačných technológií v merítku posledných desiatich rokov poukazuje na zásadné zmeny a trendy, ktorými sa riadi dnešný vývoj a používanie softvéru.

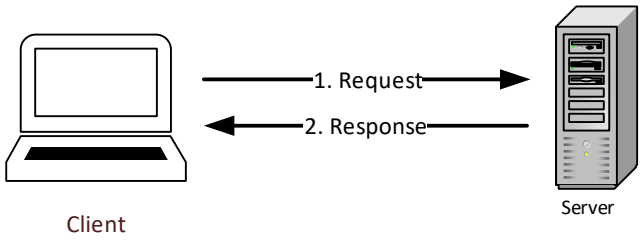
V minulosti predstavoval software vo valnej väčšine standalone aplikácie. Tieto aplikácie boli distribuované pomocou nosičov ako CD alebo DVD. Standalone aplikáciou rozumieme softvér, ktorý je nasadený, nainštalovaný na zariadenie konkrétnej platformy. K svojej funkcionalite nepotrebuje pripojenie k internetu, prístup k vzdialenej databáze a pod. Problematickou doménou je udržateľnosť z pohľadu aktualizácií. Softvér nie je riadený centrálne, ale naopak smeruje do mnohých, v určitom zmysle izolovaných smerov. Výsledkom sú vysoké finančné nároky spojené s udržateľnosťou každého typu takéhoto softvéru.

Progres v oblasti konektivity (rýchlosti a kvality pripojenia, definícii protokolov atď.) umožnil softvéru sieťovú komunikáciu, čo otvorilo nové možnosti v oblasti softvéru ako business nástroja a s tým súvisiacej komplexnosti. S vývojom konektivity implicitne vzniká model klient-server. Princípom modelu je rozdelenie (distribúcia) kompetencií medzi klientskú a serverovú časť, popisuje vzťah medzi kooperujúcimi programami v rámci aplikácie. Model môže byť nasadený v rámci jediného počítača, dôležitejším princípom je ale komunikácia cez LAN alebo WAN (internet) sieť. Klient v tomto modele komunikuje so serverom, pričom nepozná implementačné a ďalšie detaily. [19]
Klient rozumie:

- doméne úloh, ktoré server dokáže pre daného klienta vykonať
- aplikačnému protokolu, vie ako danú službu dopytovať
- odpovedi a formátu dát

Server rozumie :

- existencii klienta a doméne úloh daného klienta
- formátu komunikácie s klientom
- implementačným detailom



Obr. 24: Základná podoba architektúry typu klient-server

Rozdelenie kompetencií softvérovým jednotkám a ich existencia na rôznych zariadeniach predstavuje problematiku distribúcie aplikácie - fyzického nasadenia a logického členenia.

4.1.1 Fyzický pohľad

Fyzická architektúra špecifikuje, kde je daný kód nasadený a spustený. Fyzické vrstvy sú miesta, kde sú nasadené logické vrstvy aplikácie. Vrstvy predstavujú do určitej mieri nezávislé moduly, často nasadené na rôznych platformách.

Standalone aplikácie reprezentujú jednovrstvovú fyzickú architektúru. Klient (obsahuje logiku zobrazovania dát) ktorý komunikuje so serverom (obsahuje dáta a aplikačnú logiku) reprezentuje dvojvrstvovú architektúru. V škálovaní fyzických vrstiev ďalej hovoríme o trojvrstvovej a n- vrstvovej architektúre. Troj a n-vrstvové architektúry sú typicky používané v oblasti vývoja webových aplikácií, informačných systémov a komplexných aplikácií.

4.1.2 Logický pohľad

Logická architektúra predstavuje spôsob organizovania kódu do vrstiev vzhľadom na špecifickú funkcionalitu. Nehovorí nič o tom, kde je daná vrstva nasadená a spustená v rámci stroja alebo procesu. Typicky používané logické vrstvy v rámci trojvrstvovej logickej architektúry sú:

- prezentačná vrstva
- business vrstva
- dátová vrstva
- servisná vrstva

Všetky logické vrstvy môžu existovať v rámci klienta, ale taktiež môže byť každá logická vrstva nasadená na samostatnej fyzickej vrstve.

4.1.3 Distribúcia a nasadenie

Popísané logické a fyzické vrstvy predstavujú strategický mechanizmus pri návrhu a vývoji aplikácie. Vhodný návrh architektúry softvéru je ovplyvnený mnohými faktormi, ako :

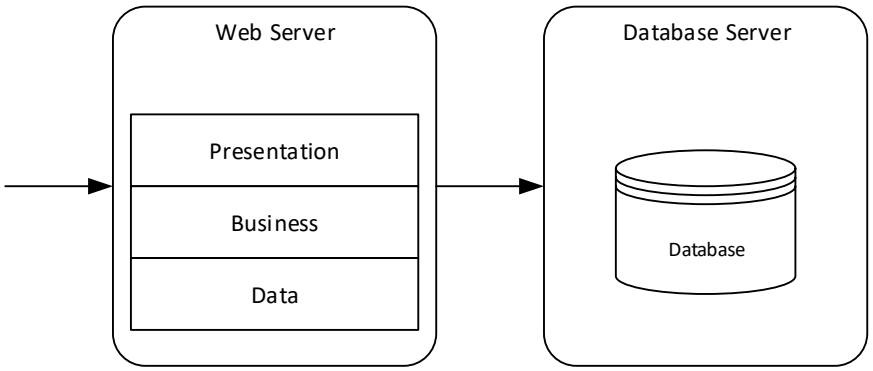
- komplexnosť aplikácie
- škálovanie aplikácie z pohľadu výkonu
- škálovanie aplikácie z pohľadu prístupu a nasadenia
- bezpečnosť aplikácie
- nezávislosť a zameniteľnosť súčastí
- migrácia súčastí aplikácie

Distribúcia a nasadenia predstavujú rozhodnutie, aké fyzické vrstvy budú v rámci aplikácie existovať a aké logické vrstvy budú na nich nasadené.

Nedistribuované nasadenie

V nedistribuovanom nasadení je všetká funkcionalita a vrstvy sústredené v rámci jedného servera, typicky je separované dátové úložisko.

Výhodou takéhoto riešenia je jednoduchosť a minimalizácia počtu fyzických serverov. Taktiež je redukovaná záťaž, keďže nie je nutné vykonávať prenosy dát medzi vrstvami po fyzickom vedení. Keďže všetky vrstvy zdieľajú zdroje, problém v rámci jednej vrstvy môže ovplyvniť aj ostatné. Toto riešenie ovplyvňuje možnosti škálovania a údržby, keďže všetky vrstvy existujú na jednom fyzickom zariadení.

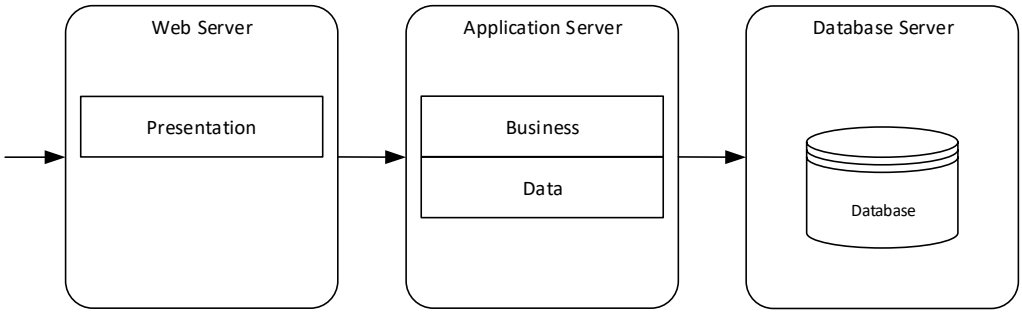


Obr. 25: Nedistribuované nasadenie [20]

Distribuované nasadenie

V distribuovanom nasadení vrstvy aplikácie existujú v rámci separátnych fyzických zariadení. Distribuované nasadenie umožňuje konfiguráciu každého fyzického zariadenia samostatne, pre konkrétne účely vrstiev ktoré sa na ňom nachádzajú. S distribuovaným nasadením prichádza možnosť voľby rozmiestnenia jednotlivých vrstiev vzhľadom na požiadavky a špecifickú funkcionalitu, alebo ich združenie v prípade potreby. Zvyšovanie úrovne distribúcie však znamená

nárast komplexnosti a ceny. Ďalším podstatným faktorom je bezpečnosť. Distribuované nasadenie umožňuje určenie bezpečnostných pravidiel v podobe firewallu, rôznych typov autorizácie a pod.



Obr. 26: Distribuované nasadenie [20]

Servisne orientované aplikácie

Servisne orientovaná architektúra predstavuje variantu modelu klient-server. Konvenčná implementácia modelu klient-server znamená, že server existuje a pracuje pre konkrétneho klienta. Hovoríme o tzv. couplingu, teda závislosti a zviazanosti týchto entít.

Servisne orientovaná architektúra je architektonický štýl budovania softvéru, princípom je minimalizácia couplingu a snaha o znovu použiteľnosť celkov. Princípom je skladanie a interakcia medzi relatívne nezávislými, bez stavovými (sieťovými, web) službami, ktoré slúžia ako mechanizmus komunikácie. V SOA je serverová funkcionalita nezávislá na klientovi. Mnoho klientov môže komunikovať s rovnakým serverom. Predpokladom je, že klient rozumie ako má s daným serverom komunikovať. Klient následne často implementuje len prezentačnú vrstvu, všetky operácie vykonáva pomocou služieb, ktoré poskytuje server. [24]

4.1.4 SOAP a WSDL

SOAP (Single Object Access Protocol) je protokol pre výmenu správ, ktoré sú založené na XML, medzi aplikáciami. Tento protokol má uplatnenie najmä v doméne webových služieb a preto sú správy najčastejšie zasielané prostredníctvom protokolu HTTP. Definuje súbor pravidiel, ktoré sú používané pre zasielanie jednosmerných správ, hlavným použitím je však vykonávanie RPC (Remote Procedure Call) založených dialógov pre požiadavku a odpoveď. Ide o platformovo nezávislú technológiu. Správa v SOAP je XML dokument, ktorý obsahuje :

- identifikácia dokumentu ako SOAP správy
- hlavička správy
- telo správy, obsahuje volanie procedúr a odpovede

- element obsahujúci chyby a stavové informácie

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Obr. 27: Ukážka SOAP požiadavky a odpovede [21]

V súvislosti s technológiou SOAP je často používaným pojmom WSDL (Web Services Description Language). WSDL je jazyk pre popis rozhrania služby. Protokolom SOAP je možné volať funkcie v zmysle služieb, WSDL popisuje čo konkrétne a ako je možné volať. WSDL popisuje názvy dostupných operácií, typy parametrov a návratových hodnôt. [23] WSDL je jazyk založený na XML, hlavnými časťami sú :

- types, definícia dátových typov
- message, komunikačná správa
- portType, súbor operácií, odpovedá definícii rozhrania
- operation, odpovedá metóde, alebo funkcii
- binding, definuje možný spôsob prístupu prístupu rôznymi protokolmi
- service a port, pre každý binding, definuje adresu s ktorou sa má príslušny protokol spojiť

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Obr. 28: Ukážka WSDL správy pre operáciu typu požiadavka-odpoveď [22]

4.1.5 REST

REST (Representational State Transfer) je architektonický vzor pre vytváranie API rozhraní, ktoré používa ako základnú komunikačnú metódu protokol HTTP (môže byť použitý aj s inými protokolmi). REST bol pôvodne koncipovaný Royom Fieldingom v roku 2000 v jeho dizertačnej práci. V súvislosti s HTTP je dôležitých niekoľko implementačných detailov :

- zdroje (resources) - REST používa adresovateľné zdroje k definovaniu štruktúry API. Jedná sa o URL, ktoré sa používajú na prístup k zdroju
- typ požiadavky (request verbs) – určujú typ operácie so zdrojom. Pre získanie, resp. čítanie zdroja je typicky používaný typ GET, ďalšie známe metódy sú POST, PUT a DELETE.
- hlavička požiadavky (request headers) – rozširujúce informácie požiadavky, ako napríklad požadovaný typ odpovede, typ autorizácie a pod.
- telo požiadavky (request body) – data zasielané spolu s požiadavkou, pre typ POST (používaný pre vytvorenie zdroja) sú v tele požiadavky samotné data
- telo odpovede (response body) – odpoveď, môže to byť HTML stránka, JSON, XML document alebo text
- kód odpovede (response status code) – kód indikujúci typ odpovede

REST je oproti SOAP orientovaný dátovo, nie procedurálne. Ide o definovanie spôsobu prístupu k dátam. V súvislosti s HTTP a dátovo orientovaným prístupom sa ustálilo používanie GET, POST, PUT, DELETE operácií pre získanie, vytvorenie, aktualizáciu a zmazanie zdroja-dát. V literatúre sa často hovorí o mapovaní CRUD požiadaviek na spomenuté HTTP metódy, exaktnejším vyjadrením je však implementácia, resp. používanie HTTP metód pre prácu zo zdrojom (mapovanie indikuje povinnosť takéhoto používania, čo nie je predmetom REST).

REST ako formát požiadaviek a odpovedí používa prevažne JSON. V porovnaní s XML je hlavnou výhodou formátu JSON redukcia veľkosti, čo znamená prenos menšieho objemu dát – pre webové služby jeden z hlavných faktorov. Použitý formát môže byť však ľubovoľný, v REST ide o dodržanie určitých princípov, nie o prevzatie konkrétnych formátov.

Základným princípom REST je bezstavovosť. Volania sú vykonávané nezávisle a každé volanie by malo obsahovať dostatok potrebných informácií a dát pre svoje vykonanie. REST API by sa z princípu nemalo spoliehať na stavové informácie servera v zmysle session. Cieľom škálovateľnosti aplikácií používajúcich REST API je ukladanie potrebného stavu u klienta, nie na servery. [25]

```
https://www.googleapis.com/language/translate/v2?key=INSERT-YOUR-
KEY&source=en&target=de&q=Hello%20world
```

Obr. 29: Ukážka volania REST API

4.2 Špecifikácia zadania

Primárnym cieľom návrhu a vývoja aplikačného rozhrania je vytvorenie servisnej vrstvy, ktorá dokáže komunikovať s klientami. Cieľom je vytvorenie vrstvy systému za účelom zvýšenia nezávislosti jednotlivých častí. Klientské aplikácie sú:

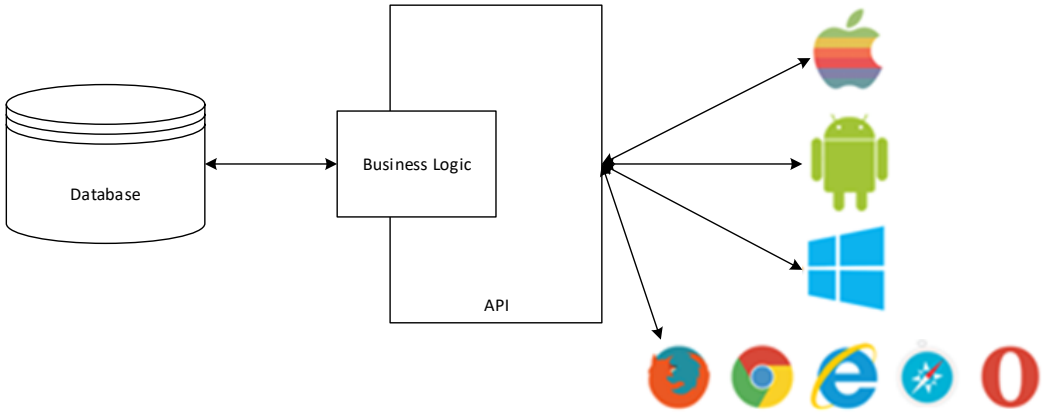
- webová aplikácia v PHP a frameworku Nette
- mobilná aplikácia vo frameworku Ionic 2

Veľmi dôležitým a zásadným faktorom je, že webová aplikácia aj mobilná aplikáciá sú privátne aplikácie v zmysle vzťahu k entite informačného systému a firmy ako celku. Vyvíjané API má služiť pre webovú stránku a mobilnú aplikáciu jednej a tej istej firmy. Webová stránka a mobilná aplikácia sa v tomto ponímaní často označujú ako „1st party software“.

Webová aplikácia v súčasnosti predstavuje klienta s kompletnou funkcionalitou. Pre vývoj API to znamená, že webová aplikácia predstavuje predpis funkčných celkov. V danej verzii vývoja sa predpokladá, že API rozhranie nebude dostupné verejne v zmysle B2B poskytovania. Klientské aplikácie sú vyvíjané v rámci privátnej infraštruktúry a cieľom vývoja je centralizovať business logiku v rámci aplikačného rozhrania. Klientské aplikácie by mali vo výsledku zastávať len prezentačnú vrstvu a interakciu s používateľom.

Predmetom implementácie API je vytvorenie funkčných blokov (po vzore webovej aplikácie), vrátane zabezpečenia a nasadenia. Samotná dátová vrstva ktorá zaobstaráva odpoveď vzhľadom na zaslanú požiadavku nie je súčasťou tejto implementácie.

Aktuálna verzia IS prezentovaná formou webovej aplikácie obsahuje verejne dostupnú doménu (stránky), doménu dostupnú po prihlásení používateľa a doménu pre administratívnych používateľov. Identita registrovaných používateľov je sústredená do tabuľky relačnej schémy, prihlásenie prebieha pomocou zadania mena a hesla. Z pohľadu klientských aplikácií a API sa jedná o centralizovaný spôsob zabezpečenia, vo vyvíjanom IS neexistuje iný spôsob overenia identity. Pre návrh a vývoj API to znamená, že zabezpečenie bude založené na overení používateľa práve vzhľadom na odpovedajúcu relačnú tabuľku obsahujúcu centralizované informácie. Administratívne typy používateľov sú analogicky implementované spôsobom pridelenia rolí danému používateľovi, ktoré sú taktiež sústredené v rámci relačnej schémy. Vývoj API pojednáva domény zabezpečenia po vzore webovej aplikácie.



Obr. 30: Diagram komunikácie API a klientov [25]

4.3 Návrh a implementácia

V časti Analýza riešení sú pojednávané dva spôsoby pomocou ktorých možno implementovať servisnú vrstvu v zmysle API. SOAP a REST nemožno porovnávať priamo, keďže sa jedná o principiálne odlišné veci - SOAP predstavuje protokol, zatiaľ čo REST architektonický štýl. Obe odpovedajú na otázku komunikácie aplikácií cez HTTP (REST aj SOAP je možné použiť mimo HTTP, táto varianta je však používaná minimálne a súvislosti s webovými službami má zmysel uvažovať výlučne protokol HTTP).

Porovnávanie SOAP a REST je veľmi často diskutovaná téma. Základný rozdiel, ktorý je relevantný pre tento návrh a implementáciu je flexibilita. SOAP je jasne určený a štandardizovaný, REST je naopak flexibilný v zmysle existencie niekoľkých princípov, ktoré je vhodné dodržiavať (nie nutné). Táto flexibilita na druhú stranu spôsobuje, že REST je často implementovaný veľmi voľne a nie sú dodržiavané základné RESTful zásady. Príkladom pre používanie SOAP sú bankové sektory, resp. prostredie kde používanie striktného protokolu dáva zmysel. SOAP z pohľadu bezpečnosti webových služieb prináša výhody použiteľné na podnikovej úrovni (atomickosť operácií, vstavaný systém na správu chýb, pokročilá dátová integrita atď.). Pre vyvíjané API je kľúčovým faktorom práve flexibilita. API v čase návrhu a vývoja nepotrebuje dodržiavať striktný protokol, ani používať spomenuté vlastnosti pre podnikové použitie. Pre vývoj API bol z dôvodu flexibility a jednoduchosti zvolený architektonický štýl REST, so snahou o maximálne dodržiavanie RESTful zásad.

4.3.1 Bezpečnosť

Infraštruktúra IS určuje spôsob a miesto uloženia registrovaných používateľských identít. V prostredí webovej aplikácie je celý proces založený na overení identity (autentifikácia) a vytvorení session na strane servera, ktorá predstavuje informáciu o prihlásení. Používateľské meno a heslo, ktoré je zasielané pri procese prihlasovania predstavuje kritickú a citlivú informáciu. Cieľom

všetkých techník, ktoré sa snažia o implementáciu zabezpečenej autentifikácie je minimalizácia zasielania a práce s citlivými a kritickými údajmi. Problematika zabezpečenia webových technológií je veľmi komplexná – zabezpečenie možno riešiť z pohľadu transformačného protokolu, aplikačného protokolu, zabezpečenie uloženia dát atď. V otázke zabezpečenia je nutné rozlišovať pojmy autentifikácia a autorizácia. Autentifikácia je overenie identity používateľa, identifikuje kto vykonáva požiadavku. Autorizácia overuje, či má vykonávateľ požiadavky dostatočné oprávnenie. V súvislosti s webovými službami existuje niekoľko hlavných a ustálených prístupov.

SSL/TLS

Ide o kryptografické protokoly, ktoré zabezpečujú šifrovanie dát a bezpečnú komunikáciu. Protokoly zabraňujú odpočúvaniu, manipulácií a falšovaniu dát. Webové technológie adaptujú tento protokol v podobe HTTPS – zabezpečeného HTTP protokolu. V implementovanom riešení sa predpokladá plné používanie HTTPS.

Basic Authentication

Predstavuje najjednoduchšiu metódu. Používa špeciálnu HTTP hlavičku, v ktorej sa nachádza meno a heslo zakódované algoritmom base64. Meno a heslo nie je kryptované a dá sa veľmi jednoducho previesť do čitateľnej podoby. Používanie sa preto neodporúča mimo zabezpečeného SSL/TLS (HTTPS) pripojenia. Nevýhodou riešenia je, že meno a heslo sa zasiela v hlavičke každej požiadavky, čo zvyšuje rámec možného napadnutia. Možným riešením je používanie HMAC (hash based message auhentication). Princípom je posielanie hashovanej verzie hesla. Typicky sa do hashovanej správy posielajú ďalšie informácie (dátum), ktoré môžu identifikovať podozrivú aktivitu.

```
GET /users/username/account HTTP/1.1
Host: example.org
Authentication: hmac username:123456:[digest]
Date: 20 apr 2013 12:59:24
```

Obr. 31: HMAC Basic autentifikácia

OAuth 1.0

OAuth (všeobecne) predstavuje otvorený štandard autorizačného protokolu. Popisuje, ako môžu servery, služby a aplikácie umožniť bezpečnú komunikáciu autentifikovaným používateľom pre prístup k dátam a zdrojom. Cieľom je eliminácia zdieľania a frekventovaného narábania s používateľskými prístupovými informáciíámi (meno a heslo). Tento štandard implementuje väčšina spoločností, ako Facebook, Google, Twitter a pod.

OAuth 1.0 vznikol v roku 2007. Považuje sa za bezpečnejší, komplexnejší, ale menej flexibilný štandard v porovnaní s OAuth 2.0. Štandard je založený na existencii zdieľaných tajných „hesiel“ medzi konzumentom a serverom, ktoré sú používané na výpočet signatúr – server overuje autenticitu požiadavky podľa signatúry. OAut 1.0 je bezpečne použiteľné aj mimo TLS/SSL. OAuth definuje spôsob autorizácie, nie autentifikácie.

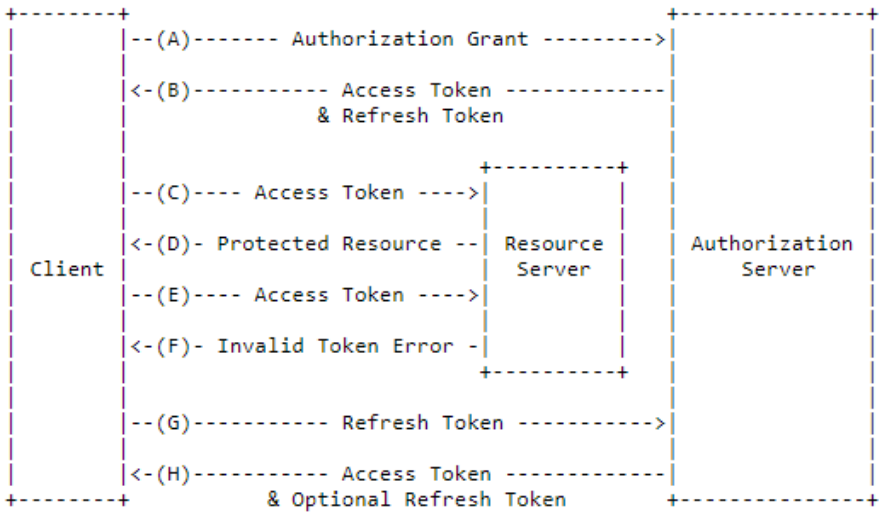
OAuth 2.0

Ide o diametrálne iné riešenie oproti OAuth 1.0. Nepoužívajú sa signatúry, naopak je vyžadované a spoliehané na SLL/TLS pripojenie (dôvod nižšej bezpečnosti). Flexibilita OAuth 2.0 spočíva v definovaných rolách a dostupných možnostiach prideľovania typov grantov. OAuth 2.0 definuje role :

- resource owner – používateľ; ten kto autorizuje aplikáciu pre prístup k účtu
- resource server – server; server ktorý hosťuje chránené dáta
- authorization server – server; server ktorý poskytuje tokeny klientovi, tokeny sú používané na prístup k resource serveru, často je resource a authorization server spojený v jednej inštancii
- client – aplikácia; aplikácia ktorá žiada prístup k resource serveru, web stránka, mobilná aplikácia atď.

Tokeny sú náhodne generované textové reťazce autorizačným serverom, ktoré sú poskytované klientovi pri požiadaní. Existujú dva typy :

- access token – umožňuje získanie dát z resource servera, token je posielaný klientom, väčšinou ako parameter, alebo hlavičky požiadavky. Má obmedzenú platnosť, ktorú definuje autorizačný server. Ide o dôvernú informáciu
- refresh token – slúži na získanie nového access tokena po jeho expirácií. Používanie tohto tokenu je závislé na implementácií.



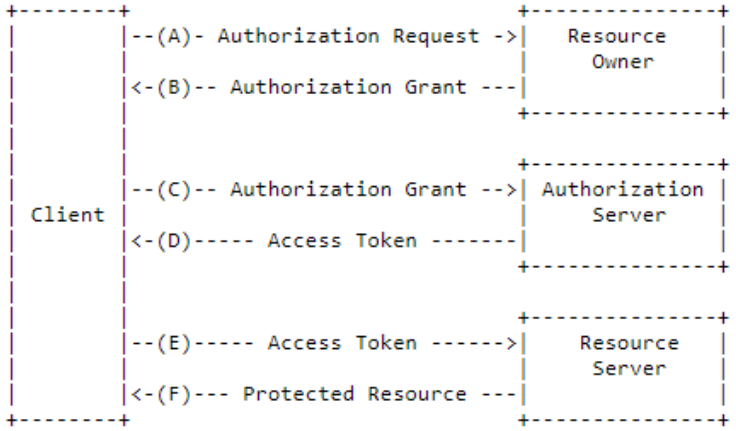
Obr. 32: Diagram toku access a refresh tokenu [26]

Primárne použitie tokenov je z dôvodu redukcie zasielania citlivých údajov v každej požiadavke. Diagram zobrazuje proces používania tokenov. Klientská aplikácia pomocou definovaného typu grantu (popísané nižšie) získa access a refresh token. Ten je následne použitý na prístup ku chránenému zdroju v rámci resource servera, ktorý vráti požadovaný obsah alebo vykáže chybu (nedostatočné oprávnenia, alebo expirovaný token, chýbajúce dáta a pod.). V prípade expirácie tokenu je použitý refresh token, pomocou ktorého je z autorizačného servera získaný nový access token. Celý proces používania refresh tokenu je nepovinný.

Ďalším definovaným pojmom je scope (access token scope) – znamená rozah práv. Scope definuje rozsah práv daného access tokenu. Zoznam a rozsah práv definuje autorizačný server. Klient typicky žiada o práva v požiadavke , ktorú prijíma autorizačný server.

Ako bolo spomenuté, flexibilita štandardu OAuth 2.0 spočíva v definvoaní typov pridelených autorizačných grantov. Každý typ má uplatnenie v iných scenároch použitia. Definované sú 4 typy :

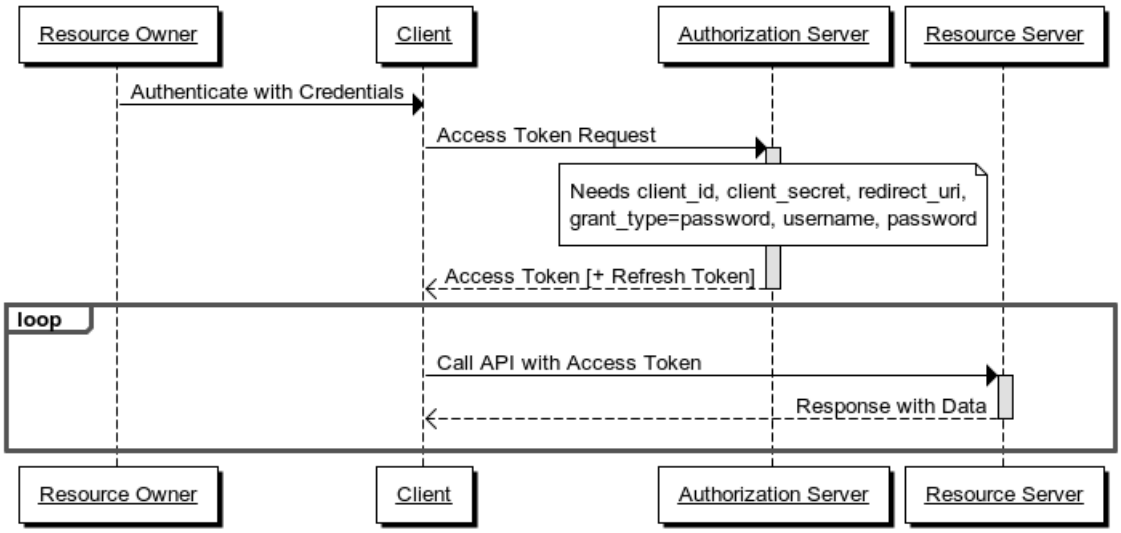
- auhorization code grant – používaný dôvernými a verejnými klientami k výmene autorizačného kódu k získaniu access tokenu
- implicit grant – zjednodušený spôsob používaný verejnými klientami, access token je vrátený okamžite bez ďalších dodatočných krokov
- resource owner password credentials grant – používaný first-party klientami (klienti jednej firmy, business modelu) k výmene používateľských prístupových práv (meno a heslo) k získaniu access tokenu
- client credentials – používaný klientami k získaniu access tokenu mimo kontext používateľa



Obr. 33: Abstraktné znázornenie toku v OAuth 2.0 [26]

OAuth definuje spôsob autorizácie, nie autentifikácie. Špeciálnym prípadom je však práve typ grantu resource owner password credentials grant. Tento grant je možné použiť ako priamy spôsob autentifikácie k získaniu access tokenu.Predpokladom pre použitie tohto grantu je veľmi vysoká dôvera medzi resource owner (používateľom) a klientom. Samotný RFC dokument popisuje, že tento spôsob je možné použiť pre first-party a vysoko delegované klientské aplikácie, čo plne vyhovuje situácií pre ktorú je API vyvíjané. Autentifikačný a autorizačný spôsob bude implementovaný spôsobom resource owner password credentials grantu.

Nasledujúci sekvenčný diagram zachytáva tento spôsob granty v rámci prvkov OAuth 2.0



Obr. 34: Sekvenčny diagra pre Resource Owner Password Credentials grant [27]

V súvislosti so zvoleným architektonickým vzorom REST a zvoleným spôsobom autentifikácie a autorizácie pomocou OAuth 2.0 Resource Owner Password Credentials grantu je nutné

uvažovať správne dodržiavanie RESTful zásad. Táto otázka je relevantná najmä k spôsobu zaobchádzania s tokenmi (access a refresh) na strane servera. OAuth nedefinuje spôsob akým s tokenmi zaobchádzať, typicky sa však po vygenerovaní token ukladá v rámci pamäte, alebo dedikovanom úložisku, ktorým môže byť napríklad databáza.

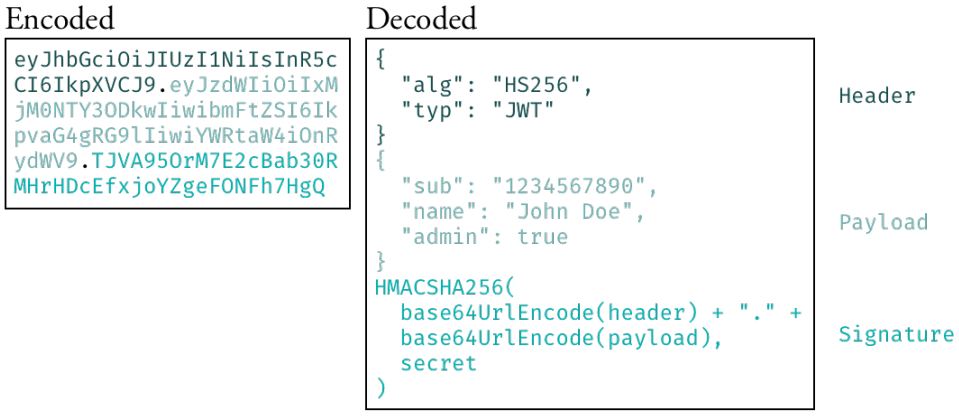
Keď následne klient zasiela požiadavku so získaným tokenom, autorizačný server musí overiť jeho legitímnosť a platnosť. Typicky je táto funkcionalita implementovaná vytváraním session, ktorá obsahuje informácie o platnosti tokenu a ďalšie informácie. Takéto riešenie porušuje princíp bez stavovosti (stateless), keďže požiadavka nenesie všetky potrebné informácie, je nutné uchovávať určitý stav na strane servera. Možným riešením sú JWT (JSON Web Token) tokeny.[27]

JWT Token

JWT Token je otvorený štandard, ktorý definuje kompaktný, samostatný, seba-popisujúci spôsob zabezpečeného prenosu informácií medzi entitami v podobe JSON objektu. Informácia v rámci tokenu je digitálne podpísaná, čo umožňuje jednoduchú kontrolu dôvernosti obsahu. JWT token môže byť podpísaný HMAC algoritmom, alebo párom verejných a privátnych RSA kľúčov. Token sa skladá z :

- hlavička – určuje typ tokenu a hashovací algoritmus
- payload – obsahuje ľubovoľné dáta, typicky oprávnenia
- signatúra – kombinuje zakódovanú hlavičku, zakódovaný obsah a tajnú časť, ktoré sú spracované zvoleným algoritmom, signatúra zabezpečuje, že obsah správy nikto žiadnym spôsobom nepozmenil

Typickým scenárom pre použitie je autentifikácia a prenášanie zabezpečených dát (z pohľadu obsahu). JWT token je možné transformovať na čitateľný text. V základnej forme dáta nie sú kryptované, ale len digitálne podpísané, takže token by nemal obsahovať citlivé informácie ako heslá a podobne. JWT token je možné aj kryptovať, takže informácie sú v nečitateľnej podobe a digitálne podpísané.



Obr. 35: JWT token v kódovanej a dekódovanej podobe

JWT Token je možné použiť ako access token a refresh token v rámci toku a výmeny informácií v OAuth 2.0. Token prináša výhodu eliminácie nutnosti vytvárania session a vôbec akéhokoľvek ukladania informácie na strane servera, čo indikuje správne dodržanie RESTful zásad. Ďalším znakom je redukcia prístupov do databázy, keďže token potrebuje všetko potrebné. Implementácia access a refresh tokenu vo formáte JWT tokenu tak predstavuje pridelenie identity, ktorá je plne postačujúca pre prístup k zdrojom.

Nevýhodou a rizikovou časťou môže byť správa pridelených tokenov. Keďže na servery neexistuje informácia v podobe session, scenár zakázania platnosti a zrušenia tokenu môže byť veľmi komplikovaný. Typickým scenárom môže byť nutnosť blokovania používateľa. V prípade prideleného a validného access tokenu vo formáte JWT neexistuje informácia na strane servera, na základe ktorej možno označiť pridelený token za neplatný. Riešením je v tomto prípade prideľovanie tokenov na krátku dobu. Vďaka krátkej platnosti je možné zakázať pridelenie nového tokenu pri použití refresh tokenu (vykonávať kontrolu vzhľadom na databázu len pri použití refresh tokenu), prípadne vynútiť opätovné prihlásenie, ktoré môže zabrániť získaniu nového tokenu. Ďalšími možnými riešeniami sú čierne listiny blokovaných používateľov, ktoré možno implementovať pomocou rýchlych cache úložísk ako Redis, prípadne vykonať kontrolu vzhľadom na databázu (výkonnostný faktor). V implementácií bude pre mobilnú aplikáciu použitý variant JWT tokenu. V prípade nutnosti je možné porušiť RESTful zásady a implementovať držanie dodatočného stavu – vyvíjané aplikácie a API sú privátneho charakteru, takže sa nejedná o porušenie žiadneho štandardu.[28]

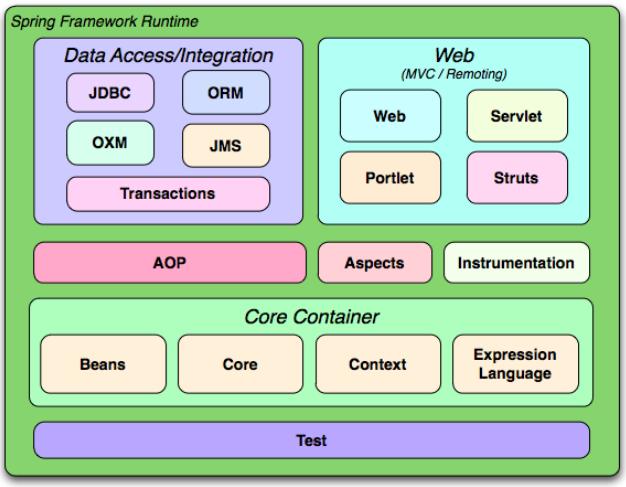
4.3.2 Spring Boot REST API

REST API s integrovaním OAauth 2.0 princípov a použitím JWT formátu tokenu môže byť implementované v rade dostupných jazykov ako PHP, C# spolu s dostupnými knižnicami a frameworkami. Zvolenou technológiou je Java z dôvodu začlenenia do back-end štruktúry ostatných aplikácií a možnosti integrácie už implementovaných častí. V rámci technológie Java je

možné implementovať popísané prvky pomocou vstavanej JSP (Java Server Pages) technológie, alebo využiť niektorý z rady dostupných webových frameworkov. Zvoleným frameworkom pre implementáciu je Spring. Spring je veľmi komplexný a modulárny framework jazyka Java, ktorý obsahuje radu dostupných funkcií ako :

- vkladanie závislostí (dependency injection)
- moduly pre objektové relačné mapovanie
- moduly pre databázové konektory
- webové moduly pre tvorbu MVC aplikácií
- moduly pre zabezpečenie aplikácií, atď.

Spring obsahuje cez 20 dostupných modulov, ktoré možno modulárne integrovať. Framework je pomerne komplexný, avšak obsahuje detailnú dokumentáciu a naprieč komunitou je široko používaný.



Obr. 36: Moduly frameworku Spring [29]

Hlavným dôvodom pre výber frameworku Spring je SpringBoot. SprinBoot možno označiť za utilitu, ktorá dokáže vytvoriť predkonfigurovanú infraštrukturálnu kostru aplikácie z modulov, ktoré ponúka Spring, nad ktorou je možné stavať vlastnú implementáciu. SpringBoot odstraňuje mnoho nutnej počiatočnej konfigurácie, takže je možné veľmi rýchlo a efektívne pristúpiť k požadovanej implementácií. V implementácií je pomocou SpringBoot kombinovaných niekoľko framework súčastí :

- Spring MVC
- Spring Security
- Spring Data JPA

- Spring Boot Starter

Pomocou Spring Boot je možné veľmi rýchlo vytvoriť kostru webovej aplikácie, ktorá komunikuje s databázou MySQL (prípadne ďalšími úložiskami), ale taktiež integruje prvky bezpečnosti na základe OAuth 2.0 štandardu (integruje samotné role). SpringBoot pre implementáciu procesu autorizácie (v konkrétnom prípade aj autentifikácie) v rámci OAuth vytvára triedy pre autorizačný server a resource server, kde je následne možné implementovať vlastný spôsob práce s tokenom, definovať platnosti tokenov a definovať zabezpečenie pre konkrétne URL, resp. domény. Vyššie spomenuté súčasti frameworku a ďalšie externé závislosti boli spracované pomocou pluginu Maven, ktorý zaobstaráva aj zostrojenie aplikácie ako celku.

V nasledujúcej ukážke je zobrazená implementácia autorizačného (v tomto prípade aj autentifikačného) servera. Prideľovanie tokenov zoabstaráva objekt userDetailsService, ktorý predstavuje mechanizmus overenia identity vzhľadom na MySQL databázu. Taktiež je tu definovaná platnosť tokenov a integrácia vlastnej token logiky – JWT token.

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
  @Bean
  public TokenEnhancer customTokenEnhancer() {
    return new CustomTokenEnhancer();
  }

  @Override
  public void configure(ClientDetailsServiceConfigurer configurer) throws Exception {
    configurer
            .inMemory()
            .withClient(clientId)
            .secret(clientSecret)
            .authorizedGrantTypes("password", "refresh_token")
        .accessTokenValiditySeconds(3600)
        .refreshTokenValiditySeconds(3600*2)
            .scopes(scopeRead, scopeWrite)
            .resourceIds(resourceIds);
  }

  /** Custom JWT token*/
  @Override
  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
    TokenEnhancerChain enhancerChain = new TokenEnhancerChain();
    enhancerChain.setTokenEnhancers(Arrays.asList(customTokenEnhancer(), accessTokenConverter));

    endpoints.tokenStore(tokenStore)
            .tokenEnhancer(enhancerChain)
            .authenticationManager(authenticationManager).userDetailsService(userDetailsService);
  }

}
```

Výpis 7: Ukážka implementácie autorizačného servera

V rámci nasledujúcej ukážky implementácie ResourceServera je zobrazené definovanie zabezpečnej domény-prefixu secure a verejnej domény-prefixu public. Taktiež je tu definovaný mechanizmus pre spracovanie tokenu, keďže pre prístup ku chránenému zdroju v rámci domény secure je nuté aby bol používateľ autentifikovaný.

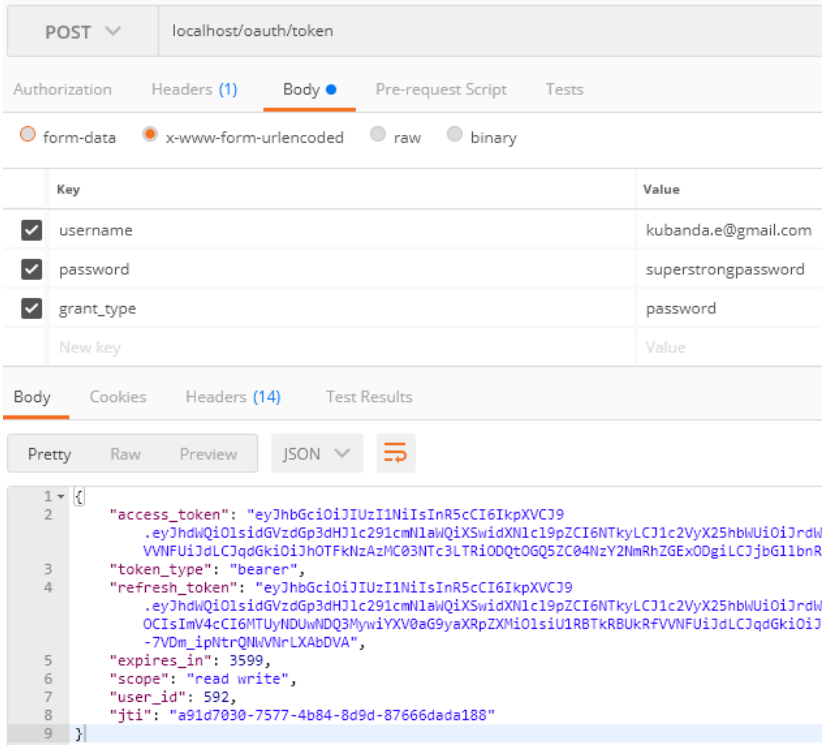
```
@Configuration
@EnableAuthorizationServer
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {
    @Autowired
    private ResourceServerTokenServices tokenServices;

    @Override
    public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
        resources.resourceId(resourceIds).tokenServices(tokenServices);
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
                http
                .requestMatchers()
                .and()
                .authorizeRequests()
                .antMatchers("/public/**").permitAll()
                .antMatchers("/secure/**" ).authenticated();
    }
}
```

Výpis 8: Ukážka implementácie resource servera

Samotné označenie, či URL spadá pod doménu public alebo secure, je definované v rámci Controllera, ktorý spracúva parametre požiadavky a buď poskytuje verejne dostupný obsah, alebo sa smeruje svoje vykonanie na definovaný proces zabezpečenia.

4.4 Testovanie a nasadenie

Aplikácia bola vyvíjaná v prostredí IntelliJ Idea verzie Ultimate. Táto verzia umožňuje spúšťať Java web-orientované aplikácie v rámci vstavaného Tomcat servera. Tento fakt podstatne zjednodušuje a urýchľuje vývoj, keďže pri testovaní nie je nutné riešiť nasadenie na externý webový Tomcat server. Samotné testovanie autentifikácie, autorizácie a dátových procesov vôbec je možné vykonať z rady nástrojov, jedným z nich je Postman. V nasledujúcej ukážke je zobrazený proces autentifikácie používateľa a úspešné obdržanie tokenov.



Obr. 37: Testovanie API z aplikácie Postman

Po testovaní a odladení aplikácie je požadované nasadiť aplikáciu na reálny server, dostupný verejne (alebo v rámci Intranetu). Pre Java web orientované aplikácie existuje rada webových a aplikačných serverov. Keďže samotné testovanie prebiehalo na vstavanom Tomcat servery, za verejne dostupný server bol zvolený taktiež Tomcat, konkrétne stabilná verzia Apache Tomcat 8. Webový server Apache Tomcat bol nainštalovaný podľa oficiálneho návodu, jedinou dodatočnou konfiguráciou bolo navýšenie kapacity pre prijímaný súbor nasadzovanej aplikácie. Pre nasadenie a distribúciu Java web orientovaných aplikácií sa používa súbor typu WAR (Web Application Archive). WAR súbor má rovnakú štruktúru ako JAR súbor. Typicky sú nasadzované v rámci servlet konrajnerov, kde sa po nasadení rozbalia a spustia. Vytvorenie a nasadenie WAR súboru aplikácie je možné vykonať pomocou nástroja Maven a odpovedajúceho pluginu. Ďalšou možnosťou je nasadenie súboru pomocou webového rozhrania Tomcat servera. Nasledujúci kód v rámci nástroja Maven vytvorí WAR súbor a inicializuje proces nasadenia.

```
clean package install tomcat7:redeploy
```

Cieľový server pre nasadenie vytvoreného WAR súboru je špecifikovaný v časti definovaných Maven závislostí v rámci pluginu tomcat7-maven-plugin.

```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
          <url>http://apitest.com:8080/manager/text</url>
          <username>root</username>
          <password>superstrongpassword</password>
          <path>/${project.artifactId}</path>
        </configuration>
      </plugin>
```

Výpis 9: Plugin pre nasadenie na vzdialený Tomcat server

Vyššie popísané procesy je možné pohodlne vykonávať z vývojového prostredia IntelliJ Idea. Proces nasadenia je možné sledovať z log záznamu, ktorý je produkovaný vo vývojovom prostredí, alebo v log záznamoch servera Tomcat. Pohodlnejšie overenie nasadenia, spustenia a stavu servera je dostupné z webového rozhrania servera Tomcat.

5 Možnosti ďalšieho rozvoja

V diplomovej práci boli pojednávané a implementované komplexné nástroje pre tvorbu fulltextovej funkcionality - Elasticsearch a API rozhrania – SpringBoot REST. Fulltextová funkcionalita a kvalitné vyhľadávacie prístupy sú predmetom dlhodobej analýzy v rámci použitého business prostredia. Skvalitnenie vyhľadávania je možné najmä v zbere používateľských dát, ktoré je možné použiť ako prídavný faktor pre výpočet skóre relevantných výsledkov. V produktovej doméne aktuálne neexistuje kvalitný synonymický slovník, jeho tvorba a integrácia by mohla rozšíriť záber vyhľadávania a vytvoriť tak intuitívnejšie vyhľadávanie. Z pohľadu škálovania výkonu je platforma pripravená, s nárastom komplexnosti však prichádzajú aj vyššie nároky na údržbu. Pri produkčnom nasadení je vhodné integrovať monitorovacie nástroje pre sledovanie dostupnosti jednotlivých prvkov.

Migrácie a dátové procesy s nárastom komplexnosti vyžadujú prepracovanejšie riešenie v spúšťaní a plánovaní, aj vzhľadom na iné (súvisiace) procesy a s ohľadom na vyťaženie častí IS.

Aplikačné rozhranie by malo postupne plne nahradiť webovú aplikáciu, predstavuje to však výrazné prepracovanie niektorých komplikovaných častí, prípadne zvolenie iného frameworku pre pohodlnejšiu a efektívnejšiu implementáciu prezentačnej vrstvy, ktorá používa API. Samostatnou kapitolou je nutnosť prepracovanejšieho spôsobu testovania, komplexnosť systému vyžaduje regresné testy pre aplikačnú logiku.

S príchodom nariadenia o ochrane osobných údajov GDPR je nutné v systéme identifikovať rizikové miesta a implementovať riešenia, ktoré zabezpečia ochranu dát.

6 Záver

Vývoj informačného systému pre reálne, produkčné nasadenie predstavuje veľmi zaujímavú, avšak komplexnú a niekedy komplikovanú prácu. Možnosť práce a vývoja s produkčnými technológiami mi priniesli cenné znalosti, ktoré okrem riešenej domény možno aplikovať aj v iných oblastiach vývoja.

Návrh a implementácia častí v produkčnej verzii prináša nové domény problémov a úloh, ktoré sú často výzvou, avšak ich riešením narastá úroveň chápania komplexných celkov. Tímový vývoj na takto komplexnom systéme považujem za jediné možné riešenie a vývojový tým hodnotím veľmi pozitívne. Pracovanie s verzovacím softvérom sa stalo nevyhnutným a bežným pri každodennom vývoji.

Implementovaný Elasticsearch cluster predstavuje kvalitný a stabilný základ. Tejto časti som venoval najviac času vzhľadom na nutnosť neustáleho testovania a hľadania ideálnych variánt. Pri vývoji API bolo komplikovanou časťou pochopenie širokého záberu bezpečnostných mechanizmov a ich správne uchopenie vzhľadom na vyvíjané riešenie. Pri vývoji API som spolupracoval s vývojárom dátovej vrstvy a konzumentom obsahu, ktorí mi poskytovali cenné podnety pre opravy a zmeny v implementácií. Získané skúsenosti a rozhľad v technológiách sú prakticky použiteľné v širokej škále ďalších úloh.

Literatúra

[1] Aruleba, Kehinde & Akomolafe, Dipo & Afeni, Babajide. (2016). A Full Text Retrieval System in a Digital Library Environment. Intelligent Information Management. 08. 1-8. 10.4236/iim.2016.81001.

[2] Full Text Search. Couchbase [online]. [cit. 2018-04-25]. Dostupné z: https://developer.couchbase.com/documentation/server/current/sdk/full-text-search-overview.html

[3] KRELLENSTEIN, Marc. Full Text Search Engines vs. DBMS. LucidWorks [online]. 2009 [cit. 2018-04-25]. Dostupné z: https://lucidworks.com/2009/09/02/full-text-search-engines-vs-dbms

[4] KRÁTKÝ, Michal a Radim BAČA. Databázové systémy [online]. [cit. 2016-04-21]. Dostupné z: http://dbedu.cs.vsb.cz/SubPages/OpenFile.aspx?file=book/dbcb.pdf

[5] GARCIA-MOLINA, Hector, Jeffrey D ULLMAN a Jennifer WIDOM. Database systems: the complete book. 2nd ed. Upper Saddle River: Pearson Prentice Hall, c2009. Pearson international edition. ISBN 9780131873254.

[6] DIETRICH, Suzanne Wagner. a Susan. URBAN. An advanced course in database systems: beyond relational databases. Upper Saddle River, N.J.: Pearson/Prentice Hall, c2005. ISBN 9780130428981.

[7] Apache Lucene 7.3.0 Documentation. Apache Lucene [online]. [cit. 2018-04-25]. Dostupné z: https://lucene.apache.org/core/7_3_0/index.html

[8] Apache Solr Reference Guide. Solr [online]. [cit. 2018-04-25]. Dostupné z: https://lucene.apache.org/solr/guide/6_6/

[9] GORMLEY, Clinton a Zachary TONG. Elasticsearch: the definitive guide. Sebastopol, CA: O'Reilly, 2015. ISBN 1449358543.

[10] Elasticsearch Reference. Elastic [online]. [cit. 2018-04-25]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/6.2/index.html

[11] SHIVAKUMAR, Shailesh Kumar. Architecting high performing, scalable and available enterprise web applications. ISBN 9780128022580.

[12] The SQL vs NoSQL Difference: MySQL vs MongoDB. Medium [online]. 2017 [cit. 2018-04-25]. Dostupné z: https://medium.com/xplenty-blog/the-sql-vs-nosql-difference-mysql-vs-mongodb-32c9980e67b2

[13] MONK, Paul. TO SQL OR NOT TO SQL. Capgemini [online]. 2017 [cit. 2018-04-25]. Dostupné z: https://capgemini.github.io/design/sql-vs-nosql/

[14] The JSON Data Type. MySQL [online]. [cit. 2018-04-25]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/json.html

[15] MongoDB Architecture. MongoDB [online]. [cit. 2018-04-25]. Dostupné z: https://www.mongodb.com/mongodb-architecture

[16] Why CouchDB? [online]. [cit. 2018-04-25]. Dostupné z: http://docs.couchdb.org/en/2.1.1/intro/why.html

[17] Couchbase and Apache CouchDB compared. Couchbase [online]. [cit. 2018-04-25]. Dostupné z: https://www.couchbase.com/couchbase-vs-couchdb

[18] Jenkov [online]. 2014 [cit. 2016-04-15]. Dostupné z: http://tutorials.jenkov.com/java-utilconcurrent/ threadpoolexecutor.html

[19] SHKLAR, Leon. a Rich. ROSEN. Web application architecture: principles, protocols and practices. 2nd ed. Hoboken, NJ: Wiley, c2009. ISBN 047051860x.

[20] Chapter 19: Physical Tiers and Deployment. Microsoft [online]. [cit. 2018-04-25]. Dostupné z: https://msdn.microsoft.com/en-us/library/ee658120.aspx

[21] XML Soap. W3schools [online]. [cit. 2018-04-25]. Dostupné z: https://www.w3schools.com/xml/xml_soap.asp

[22] XML WSDL. W3schools [online]. [cit. 2018-04-25]. Dostupné z: https://www.w3schools.com/xml/xml_wsdl.asp

[23] WEERAWARANA, Sanjiva. Web services platform architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and more. Upper Saddle River, NJ: Prentice Hall PTR, c2005. ISBN 0131488740.

[24] R. KODALI, Raghu. What is service-oriented architecture?: An introduction to SOA. Javaworld [online]. 2005 [cit. 2018-04-25]. Dostupné z: https://www.javaworld.com/article/2071889/soa/what-is-service-oriented-architecture.html

[25] KEARN, Martin. Introduction to REST and .net Web API. Microsoft [online]. 2015 [cit. 2018-04-25]. Dostupné z: https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

[26] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012. Dostupné z: http://www.ietf.org/rfc/rfc6749.txt

[27] REINKE, Johann. Understanding OAuth2. BubbleCode [online]. 2016 [cit. 2018-04-25]. Dostupné z: http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/

[28] Introduction to JSON Web Tokens. JWT [online]. [cit. 2018-04-25]. Dostupné z: https://jwt.io/introduction/

[29] Introduction to Spring Framework: Overview of Spring Framework [online]. [cit. 2018-04-25]. Dostupné z: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html

A Príloha na CD/DVD

Súčasťou diplomovej práce je CD/DVD médium, ktorého obsahom je: